VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

INOVACE SYSTÉMU PRO AUTOMATIZOVANÉ INTEGRAČNÍ TESTY ELEKTRONICKÝCH JEDNOTEK VOZIDEL

INOVATION OF SYSTEM FOR AUTOMATED INTEGRATIONAL ECU TESTS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. JOSEF KYLOUŠEK**
AUTHOR

VEDOUCÍ PRÁCE **doc. Ing. MARTIN DRAHANSKÝ, Ph.D.**
SUPERVISOR

BRNO 2017

Zadání diplomové práce/19319/2016/xkylou00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2016/2017

Zadání diplomové práce

Řešitel: **Kyloušek Josef, Bc.**
Obor: Počítačová grafika a multimédia
Téma: **Inovace systému pro automatizované integrační testy elektronických jednotek vozidel**
Inovation of System for Automated Integrational ECU Tests
Kategorie: Softwarové inženýrství

Pokyny:

1. Prostudujte a analyzujte testovací nástroje používané v automobilovém průmyslu (TestAut 2, Modena, Exam, PROVEtech ...).
2. Vyberte vhodné principy a vlastnosti pro nasazení na integrační testy řídicích jednotek vozů ve Škoda Auto a.s.
3. Navrhněte způsob, jakým vybrané principy zakomponovat do nástroje TestAut 2.
4. Implementujte tato vylepšení do nástroje TestAut 2.
5. Vaše řešení otestujte a diskutujte dosažené výsledky.

Literatura:

- MEDEIROS DE CAMPOS, José Carlos, et al. Continuous test generation: enhancing continuous integration with automated test generation. In: *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*. ACM, 2014. s. 55-66.
- STAHL, Daniel; BOSCH, Jan. Modeling continuous integration practice differences in industry software development. *Journal of Systems and Software*, 2014, 87: 48-59.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Drahanský Martin, doc. Ing., Dipl.-Ing., Ph.D.**, UITS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 24. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Tato diplomová práce se zabývá problematikou softwarových nástrojů pro automatizované integrační testy elektronických jednotek vozidel. Konkrétně je zaměřena na nástroj TestAut 2, který je vyvíjen společností e4t a využíván pro testování vozů značky Škoda. Cílem této práce je porovnání tohoto nástroje s podobnými produkty v automobilovém průmyslu. Práce tedy obsahuje analýzu stávajícího nástroje TestAut 2 a popis principů a vlastností nástrojů EXAM, MODENA a PROVEtech. Z těchto získaných informací je vybrána množina principů považovaných za výhodné. V další části práce je popsána implementace těchto principů do nástroje TestAut 2, testování jejich funkčnosti a zhodnocení jejich přínosu. Závěrem je navrhnuto další vylepšení upravovaného nástroje.

Abstract

This master's thesis deals with the issue of software tools for automated integrational ECU testing. It is specifically focused on TestAut 2 tool, which is developed by e4t company and used for testing of Skoda cars. Goal of this thesis is comparison of TestAut 2 against similar tools used in automotive industry. This work contains analysis of TestAut 2 tool and description of principles and features of EXAM, MODENA and PROVEtech tools. Set of principles considerd as advantageous is chosen from collected information. Next part of this thesis describes implementation of chosen principles to TestAut 2 tool, verification of it's functionality and evaluation of it's benefits. In the conclusion author suggests another improvements that can be done.

Klíčová slova

Elektronická řídicí jednotka, ECU, integrační testování, automatizované testování, elektronika vozu

Keywords

Electronic Control Unit, ECU, Integration testing, Automated testing, Car electronic

Citace

KYLOUŠEK, Josef. *Inovace systému pro automatizované integrační testy elektronických jednotek vozidel*. Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Drahanský Martin.

Inovace systému pro automatizované integrační testy elektronických jednotek vozidel

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana doc. Ing., Dipl.-Ing. Martina Drahanského, Ph.D. Uvedl jsem všechny prameny a publikace, ze kterých jsem čerpal.

.....
Josef Kyloušek
23. května 2017

Poděkování

Chtěl bych velmi poděkovat panu doc. Ing., Dipl.-Ing. Martinu Drahanskému, Ph.D. za příkladné vedení diplomové práce a za velkou trpělivost a pochopení, které během vzniku této práce prokázal. Děkuji také svému kolegovi, panu Ing. Janu Brabcovi, za přínosné a inspirativní konzultace.

Obsah

Kapitola 1

Úvod

Pokrok nezastavíš. Pravdivost tohoto výroku neznámého autora můžeme každý den sami pozorovat ve svém okolí. Jedním z odvětví, ve kterém se pravidelně objevuje velké množství nových technologií je automobilový průmysl. Ať už se jedná o zvyšování efektivity motorů, snižování spotřeby paliva a produkce emisí škodlivých plynů, vývoj alternativních pohonů, zvyšování počtu komfortních prvků vozidla a v neposlední řadě velmi aktuální rozvoj asistenčních a takzvaně inteligentních funkcí vozu. Tento rozmach přináší velké množství nových prvků do výbavy vozu. Všechny tyto elementy musejí být řízeny a navzájem spolu komunikovat. V dnešní době je tato součinnost zajišťována ve většině případů elektronicky. Některé části elektronického systému vozidla jsou kritické pro jeho bezpečný chod. Jedná se například o správné řízení motoru, brzdných systémů, airbagů či asistenčních systémů zabraňujícím smyku vozidla. Ale i zdánlivě méně důležité prvky výbavy vozu mohou svojí chybou způsobit nehodu vozidla. Například samovolné rozsvícení osvětlení interiéru může za jízdy v noci nebezpečně snížit viditelnost pro řidiče. Je tedy velmi důležité, aby všechny prvky výbavy vozu pracovaly správně. V této práci se zaměřím na ty, které jsou ovládány elektronicky. Správná funkčnost elektroniky vozu je zajišťována vhodným návrhem, provedením a testováním jednotlivých komponent i jejich správným propojením a vzájemnou komunikací. K tomuto účelu existují normy, které musejí výrobci automobilů dodržovat.

Právě testováním komunikace jednotlivých prvků elektronického vybavení vozu se zabývá oddělení společnosti e4t electronics for transportation s.r.o. Pro tyto účely je využíván testovací nástroj TestAut 2 vyvíjený přímo v rámci společnosti. Tento nástroj má ale řadu nedostatků, které snižují jeho efektivitu a použitelnost. Proto existuje zájem tento nástroj inovovat a vyřešit tak některé známé nedostatky a přidat navíc nové funkce, které zvýší efektivitu práce s tímto nástrojem. Poté co bude v kapitole 2 vysvětlen význam některých pojmů důležitých pro tuto práci, se v kapitole 3 nachází analýza tohoto, ale i dalších nástrojů používaných pro testování v automobilovém průmyslu. Na základě získaných informací jsou v kapitole 4 prezentovány vybrané principy vhodné pro zakomponování do stávajícího systému, aby tak spojoval výhody jednotlivých zkoumaných systémů. Následuje kapitola 5 popisující návrh postupu potřebného k dosažení požadovaných vylepšení tohoto systému. V kapitole 6 je zachycen průběh implementace zvolených části systému dle předem definovaného návrhu. Jsou zde rozebrány problémy, které při tvorbě vznikly, spolu s popisem jejich řešení. A konečně kapitola 7 obsahuje postup a výsledky prováděných testů a jejich interpretaci od jednotkových testů jednotlivých komponent až po kompletní integrační test při nasazení nové verze testovacího systému do praxe.

Kapitola 2

Stanovení základních pojmů

Tato kapitola obsahuje popis několika druhů zařízení a principů důležitých pro správné a jednotné pochopení obsahu této práce. Znalosti potřebné k sepsání této kapitoly jsem částečně získal během studia na vysoké škole - jedná se o problematiku testování. Další část znalostí jsem načerpal během prvních týdnů a měsíců svého působení ve společnosti e4t - během zaučovacího procesu. V některých případech může existovat více možností, jak si daný pojem vysvětlovat. V těchto situacích volím interpretaci, která se používá v rámci naší společnosti.

2.1 Elektronika vozu

2.1.1 Řídicí jednotka

Kromě firemních zdrojů jsem pro sepsání této podkapitoly použil článek z knihy [38]. Pro označení elektronické řídicí jednotky se často používá zkratka ECU z anglického sousloví Electronic Control Unit. V této práci je dále pro řídicí jednotku používána česká zkratka ŘJ.

ŘJ je vestavěný systém určený pro ovládání a řízení funkcí automobilu. Na základě aktuálního stavu ovládaného systému vypočítá odpovídající reakci a tu provede. Jeho vstupy a výstupy jsou ve formě elektrických signálů. Proto musí existovat snímače, které potřebné veličiny v podobě tlaku, rychlosti, teploty a dalších převede na hodnoty elektrické. Na výstupy ŘJ jsou připojeny akční členy, které jsou ovládány elektrickými signály a provádějí požadované úkony. ŘJ se skládá z hardwarové a softwarové části. Hardware je většinou ve formě plošného spoje, který obsahuje jednotlivé součástky a je chráněn kovovou schránkou. Kromě již zmíněného rozhraní obsahuje také mikroprocesor, který provádí výpočty pomocí definovaných funkcí. Další podstatnou částí je pamět (nejčastěji typu EPROM nebo flash) pro uchování softwaru (firmwaru). Tento software se po nahrání do paměti (takzvaném "flashování") používá pro výpočty ŘJ. Lze jej v případě potřeby přepsat jiným (například novějším) softwarem. Kromě toho se ŘJ takzvaně "kóduje". To znamená, že se do ní ukládá informace o výbavě vozidla a tedy určuje jaké funkce má ŘJ poskytovat a jaké ne.

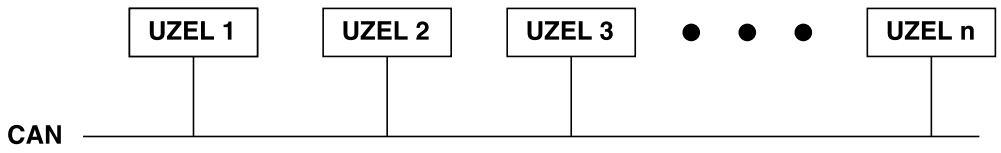
Před vznikem ŘJ probíhaly veškeré procesy v automobilu mechanicky, hydraulicky, pneumaticky, elektricky či jiným způsbem, nikoliv však elektronicky. První využití elektroniky pro řízení procesů v automobilu bylo testováno v osmdesátých letech minulého století. Použité procesory většinou nepočítaly reakci ŘJ pomocí funkcí, ale využívaly vstupní signály pro nalezení odpovídající reakce v předem definovaných vyhledávacích tabulkách. První oblastí, kde byly ŘJ používány, bylo nejsložitější místo tehdejších vozů - motor. Řízení

funkce motoru tak mohlo probíhat na základě většího množství vstupních dat a jejich porovnávání a vyhodnocování mohlo probíhat složitěji a programovatelně. Tento přístup se časem osvědčil. Přinesl zvýšení efektivity motoru, snížení spotřeby paliva a snížení emisí výfukových plynů. Díky výhodám, které použití ŘJ přináší, vznikaly postupem času ŘJ pro další oblasti automobilu. Mohly tak vznikat funkce a vylepšení automobilu, které by bez elektronického řízení nebyly možné. Současné automobily běžně obsahují desítky ŘJ. Protože funkce některých z nich je pro automobil kritická, a některé naopak slouží jen pro komfort či zábavu posádky, nenachází se zpravidla všechny řídicí jednotky na jedné sběrnici. Například informace o stavu autorádia tak nezahlcují důležitou sběrnici přenášející informace o řízení motoru či brzd. Automobil tak typicky obsahuje několik sběrnic. Každá z nich sdružuje ŘJ pro určitou oblast automobilu. Sběrnic existuje několik druhů. V automobilech společnosti Škoda Auto se používá především sběrnice CAN a pro některé účely i sběrnice LIN. Dále se v automobilovém průmyslu často využívají sběrnice FlexRay, MOST či ethernet. Jejich použití se liší podle požadovaných vlastností. Jak se píše na webu společnosti softing [3]: LIN (Local Interconnected Network) je levnější sběrnice používaná především pro vytváření malých podsítí. Například pro připojení snímačů nebo pro připojení vedlejší ŘJ k hlavní jednotce. MOST (Media Oriented System Transport) slouží především pro snadnější integraci ŘJ poskytujících informace a zábavu. K tomuto účelu má speciální komunikační mechanismus a velkou přenosovou rychlost. FlexRay je jedním z novějších typů sběrnic. Vznikl na začátku tohoto století a vyniká vysokou mírou bezpečnosti přenosu. Je tak vhodný pro kritická místa systému. V posledních letech začíná do oblasti komunikace řídicích jednotek i ethernet. Tento standard je léty prověřený použitím v počítačových sítích a nabízí levné komponenty a extrémně velké přenosové rychlosti v porovnáním s ostatními sběrnicemi. Sběrnice CAN je popsána v následující podkapitole.

2.1.2 CAN sběrnice

Pro hlubší pochopení problematiky spojené s CAN sběrnicemi jsem použil knihu popisující tuto oblast [36].

Některé informace byly použity v tomto článku. CAN (Controller Area Network) je sériový komunikační protokol vhodný pro distribuované řízení v reálném čase. Zajišťuje vysokou úroveň bezpečnosti a rychlosti pro použití v kritických oblastech jako je například propojení ŘJ motoru s dalšími důležitými jednotkami. Zároveň je ale dostatečně finančně nenáročná pro použití v oblastech méně důležitých. Topologie zapojení uzlů na sběrnici CAN je znázorněna na obrázku 2.1. Každý uzel má pouze jedno rozhraní a tím je připojen na společné vodiče. Počet připojených uzlů je omezen pouze fyzickými vlastnosti použitého média. Pokud je nějaký uzel připojen k více sběrnicím zároveň, označuje se jako "gateway"a umožňuje přeposílat vybrané zprávy z jedné sběrnice na sběrnici jinou. Kromě toho může gateway uzel například sledovat aktivitu na jedné sběrnici a po nějakém čase poslat souhrn informací v jedné zprávě na jinou sběrnici.



Obrázek 2.1: Zapojení uzlů na sběrnici CAN.

CAN byl vytvořen německým výrobcem automobilových řídicích systémů, společností Bosch, v roce 1985. O rok později byl prezentován na významném mezinárodním kongresu SAE (Society of Automotive Engineers) [39]. V roce 1991 vyšla druhá verze tohoto protokolu [35], která zachovává zpětnou kompatibilitu a například dovoluje zvětšit délku identifikátoru zprávy z 11 bitů na 29 bitů. Na základě této specifikce vznikla v roce 1993 norma ISO 11898. Jeho topologie nahradila do té doby běžné propojení, kde byl uzel připojen ke každému dalšímu uzlu samostatným vodičem. Zapojení sběrnice CAN dovoluje komunikaci všech připojených uzlů při výrazně menším počtu vodičů, čím šetří náklady a snižuje hmotnost vozu.

Informace jsou po sběrnici přenášeny ve formě zpráv. Existují čtyři druhy zpráv a každý z nich přesně definuje formát zprávy. Uzly nepotřebují znát žádnou informaci o počtu, stavu či rozmístění ostatních uzlů. To přináší několik výhod. Systém je tak flexibilní. Uzly mohou být připojovány a odpojovány, aniž by tato informace musela být šířena do ostatních uzlů. Zpráva se skládá z několika částí. Na prvním místě v pořadí je identifikátor. Ten označuje obsah zprávy. Nedefinuje její cíl, ani zdroj, ale její význam. Každý uzel pozná, zda je zpráva určena jemu podle toho, zda má definováno zpracovávat zprávy s tímto identifikátorem. CAN funguje z principu jako multicast. Je garantováno, že zprávu buď mohou číst všechny připojené uzly, nebo žádný z nich. Tím je zaručena konzistence dat mezi uzly.

Identifikátor zároveň slouží jako definice priority zprávy. Když na sběrnici neprobíhá žádná komunikace, každý z připojených uzlů může začít vysílat novou zprávu. Pokud dva nebo více uzlů začne ve stejný okamžik vkládat na sběrnici různé zprávy, bude na sběrnici vložena pouze zpráva, která má z nich nejvyšší prioritu. Princip rozhodování zaručuje, že nedojde ke ztrátě informací ani času. Funguje tak, že všechny uzly vkládající zprávu na sběrnici po vložení každého bitu kontrolují, zda se na sběrnici nachází stejná logická hodnota, kterou vkládaly. Pokud ano, pokračují dále ve vkládání. Pokud ne, znamená to, že nějaký jiný uzel vložil dominantní logickou hodnotu, která značí identifikátor s vyšší prioritou. V takovém případě uzel přestane vkládat další bity a nechá vysílat prioritnější zprávu. Může se pokusit o znovuodeslání své zprávy při další příležitosti volné sběrnice.

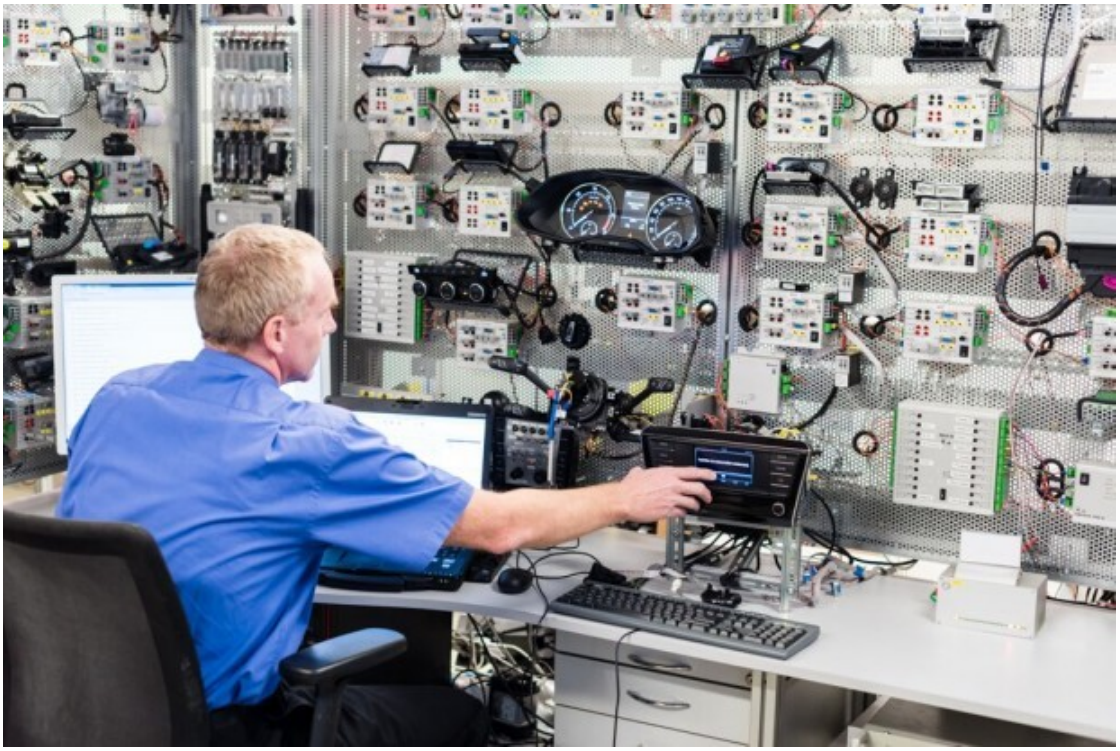
Která bitová hodnota je dominantní a která je naopak recesivní je určeno médiem, na kterém komunikace probíhá. V případě dvouvodičového vedení používaného ve vozech Škoda je dominantní hodnotou logické nula a recesivní hodnotou logická jednička. Pokud tedy ve stejný okamžik začne vysílat například uzel A zprávu s identifikátorem 00110100000, uzel B začne vysílat zprávu s identifikátorem 01010000000 a uzel C zprávu s identifikátorem 10001000000, pak při odesílání prvního bitu jednotka C detekuje, že je na sběrnici vkládána zpráva s vyšší prioritou a okamžitě přestane vysílat. Stejně zareaguje i jednotka B při odesílání druhého bitu. Jednotka A odvysílá celou zprávu.

Bezpečnost přenosu informace je zajišťována hned několika způsoby. Detekce chyb probíhá pomocí cyklického redundantního součtu a metody zvané "bit stuffing". Při detekování poškozené zprávy je každý uzel, který tuto chybu zjistil, povinen zprávu označit a její platnost je zrušena a musí být odvysílána znovu. Pokud některý uzel generuje větší množství chyb, je mu odebrána možnost vkládat zprávy a může je jen pasivně číst.

2.1.3 Breadboard

Testování řídicích jednotek probíhá na hardwarové sestavě nazývané "breadboard". Do češtiny se toto slovo překládá jako "nepájivé pole". Tímto pojmem jsou většinou označovány malé destičky obsahující v mřížce velké množství otvorů pro připojování elektrických součástek. Slouží v sestavování elektrických obvodů bez nutnosti pájet. Breadboardy používané

k testování ŘJ se od těchto destiček svým vzhledem velmi liší, ale jejich funkce je v podstatě totožná. Ukázka breadboardu je na obrázku 2.2.



Obrázek 2.2: Breadboard [31].

Jeho účelem je propojit všechny ŘJ automobilu do společné sítě, stejně jako je tomu v reálném voze. Vstupy ŘJ jsou realizovány stejným hardwarem jako v automobilu (například páčky pod volantem) nebo jsou ve zjednodušené formě přepínačů (zavření/otevření dveří) a otočných tlačítek (hladina paliva v nádrži). Některá složitá zařízení, jako například motor, která generují velké množství signálů a také přijímají mnoho ovládacích příkazů, jsou nahrazena Hardware-in-the-Loop simulátorem. To je zařízení, které přijímá signály od ŘJ, vypočítává jak by se choval skutečný motor a podle toho nastavuje hodnoty signálů simulujících snímače.

Každá ŘJ je na CAN sběrnici připojena přes zařízení zvané "modul", které lze ručně i pomocí počítače ovládat. Může odpojit či připojit ŘJ od sběrnice a napájení, provést odpojení či naopak zkrat vodičů sběrnice, měřit elektrický proud a napětí a získané hodnoty ukládat na paměťovou kartu.

Kromě již zmíněných ŘJ, modulů a HiL simulátorů se na breadboardu nacházejí ještě další prvky - zdroj elektrického proudu pro moduly; zdroj elektrického proudu pro ŘJ; elektrická zátěž připojená k tomuto zdroji; zařízení Tedia pro analogové ovládání této zátěže pro rychlejí změny napětí při simulování startovacích pulzů; diagnostická hlava umožňující vyčítání diagnostických informací z ŘJ a posledním prvkem je připojený počítač s nainstalovaným nástrojem TestAut, který spouští integrační testy.

2.2 Integrační testování

Integrační testování je jednou z částí celého testovacího procesu. Většinou následuje po jednotkových testech komponent. Tyto procesy jsou obecné, nezáleží tedy zda je komponentou myšlena funkce, třída či zařízení. Dá se aplikovat na každou z těchto úrovní. Jednotkové testy odladily chyby, které se mohou vyskytovat ve vnitřní funkci komponenty. Pokud má komponenta nějaké rozhraní pro komunikaci s okolním prostředím, jsou v jednotkových testech na tato rozhraní vkládány hodnoty uměle, ne od skutečných komponent. Právě až při testech integračních jsou jednotlivé komponenty skutečně připojeny k sobě a je jim umožněna komunikace. V těchto případech dochází někdy k chybám sporadickým, tedy i pokud se integrační test spustí několikrát se stejným počátečním nastavením, nemusí být všechny výsledky stejné. Kromě testování komunikace mezi jednotlivými komponentami lze testovat například komunikaci mezi komponentou a systémem, ve kterém je použita, nebo softwarovou komponentou a hardwarem se kterým komunikuje. Integrace většinou probíhá postupně. Tedy, že se nejprve testuje menší množství vzájemně spolupracujících komponent a další se postupně přidávají. Protože se pracuje se skutečnými komponentami, může být někdy složité dosáhnout požadovaného chybového stavu. Narozdíl od jednotkových testů je nutné, aby všechny testované komponenty již byly plně vyvinuty a otestovány. Samotný integrační test se většinou zaměřuje na dvě věci. Zaprvé kontroluje obsah zpráv posílaných mezi komponentami a porovnává je s očekávanými. Zadruhé sleduje chování komponent a jejich vzájemnou interakci. Integrační testy mohou být ruční, automatizované i kombinované.

Kapitola 3

Analýza testovacích nástrojů používaných v automobilovém průmyslu

3.1 TestAut 2

3.1.1 Historie

První verze softwarového nástroje TestAut vznikla na základě požadavků od společnosti Škoda Auto a.s., která si jeho vytvoření u společnosti e4t objednala pro potřeby integračního testování elektronických jednotek automobilu. Mezi hlavními požadavky bylo, aby testy a parametry testů byly ukládány do XML (*Extensible Markup Language*) souboru, a aby testy byly psány v programovacím jazyku Tcl [28]. Oba tyto požadavky byly splněny. První verze TestAutu byla celá napsaná v jazyce Tcl, a grafické uživatelské prostředí bylo vytvořeno s pomocí toolkitu Tk, který je s Tcl spjatý. V praxi se ale ukázalo, že toto řešení není výhodné z důvodu pomalé odezvy grafického prostředí. Proto vznikla druhá verze programu, kde byly testy ponechány v jazyce Tcl a jádro programu je psané v jazyce C++. Grafické uživatelské prostředí je vytvořeno s využitím knihovny MFC (*Microsoft Foundation Class*) [9].

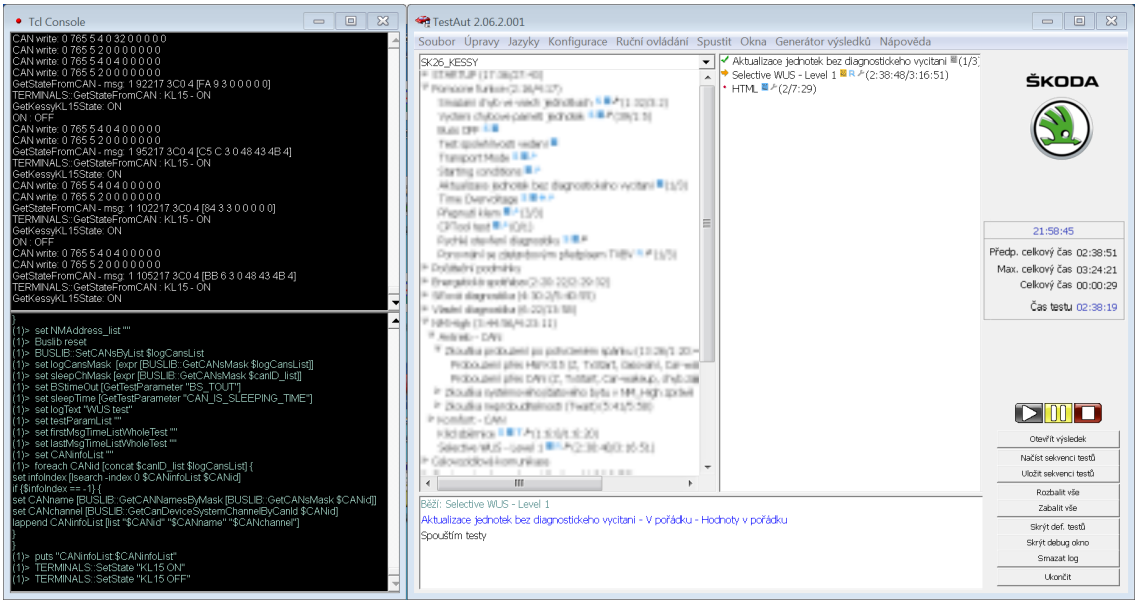
3.1.2 Tcl

Název programovacího jazyka Tcl vznikl jako zkratka anglického slovního spojení Tool Command Language. Jedná se o interpretovaný imperativní skriptovací jazyk, který je k účelům testování (ať už hardware nebo software) vhodný. Bývá dokonce označován jako průmyslový standard v prostředí automatizovaného testování [29]. Silnou stránkou je jeho snadná přenositelnost napříč platformami a jeho snadná rozšiřitelnost a propojitelnost s dalšími programovacími jazyky. Je specifický svojí jednoduchostí, která jej činí mocným nástrojem, ale v některých případech tím jeho použití komplikuje, například při vytváření složitějších konstrukcí. Mezi vlastnosti, které dovolují velmi efektivní práci patří skutečnost, že veškeré operace prováděné v tomto jazyce jsou funkcemi. Například i klasická "for"konstrukce pro provádění kódu ve smyčce je funkcí, která přijímá 4 parametry (inicializační kód, podmínku, kód prováděný na konci smyčky a samotné tělo smyčky). A stejně jako všechny ostatní funkce může být i za běhu programu předefinován! Další podstatnou vlastností jazyka Tcl je to, že vše lze zadat ve formě řetězce, a to platí pro všechny datové typy i

samotný zdrojový kód. Dovoluje tak vytvářet funkce za běhu programu. Za nevýhody jazyka Tcl považuji například dlouho chybějící funkcionalitu objektové orientace, která byla možná pouze pomocí různých rozšíření. Od verze 8.6 je součástí základní distribuce, ale stále neposkytuje všechny výhody, které objektový systém může přinést.

3.1.3 Grafické uživatelské prostředí

Na obrázku 3.1 je ukázka okna aplikace. Mezi hlavní části patří podokno "Definice testů". Nachází se v levé části hlavního okna. Jak je z názvu patrné, v definici testů je zobrazena stromová struktura obsahující všechny testy načtené z XML. Testy a sekvence testů je zde možné otevírat, upravovat, přidávat, přesouvat či mazat. Zároveň je možné z tohoto podokna přesouvat technikou "drag and drop"testy či celé sekvence testů do podokna vedlejšího, s názvem "Testovací sekvence". V této další části okna jsou zobrazeny testy připravené ke spuštění. Dalším způsobem jak přidávat testy ke spuštění je načtením uložené sekvence testů. Pod oběma podokny se nachází podokno další, které slouží jako log pro zobrazení informace o aktuálním stavu probíhajícího testu. V pravé části hlavního okna se nachází informace o době běhu testovací sekvence, ovládání průběhu sekvence s možností pozastavení či úplného zastavení a dále tlačítka pro obsluhu programu. V horní části okna je standardní lišta menu pro přístup k nastavení konfigurace programu a spouštění dalších funkcí. Pod touto lištou se nachází vysouvací nabídka, kterou se lze přepínat mezi zobrazením definice testů lokálních a serverových. Kromě tohoto hlavního okna je součástí standardního zobrazení ještě okno konzole skládající se ze dvou částí. Horní část slouží jako textový výstup spouštěných testů a dolní část vypisuje prováděné Tcl příkazy.

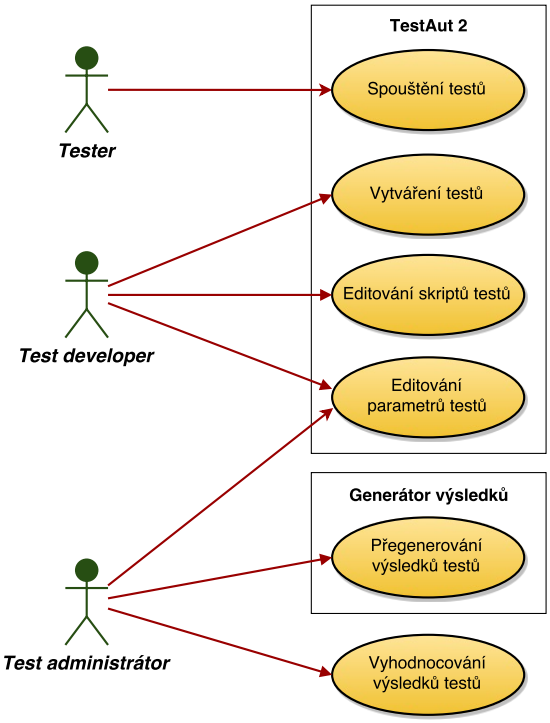


Obrázek 3.1: Ukázka grafického prostředí programu TestAut.

3.1.4 Případy použití

Některé z popisovaných operací je možné provádět pouze v takzvaném "Administrátorském režimu". Tímto způsobem se rozděluje práce s programem do různých rolí. Diagram použití

je znázorněn na obrázku 3.2. Z něj vyplývá, že tester nepotřebuje ke své práci například editovat testy, a tak je možné tyto operace provádět pouze v Administrátorském režimu. Přepnutí do tohoto režimu není chráněno heslem, lze jej vyvolat v liště menu a slouží pouze jako ochrana proti neúmyslným chybám testerů. Pracovní náplní testera je spouštět hotové a nakonfigurované testy a nastavit hardware breadboardu podle aktuálních požadavků, aby všechny testy proběhly v pořádku. Test developer vytváří testy (píše skripty a vytváří parametry) dle zadané specifikace. Pracuje většinou v administrátorském režimu. Spouštění testů provádí pouze pro potřeby ladění vyvíjených testů. Dříve než se dokončený test dostane k testerovi, musí Test administrátor správně nastavit hodnoty parametrů a vytvořit lokální potomky testu pro jednotlivé breadboardy. Taktéž pracuje především v administrátorském režimu.



Obrázek 3.2: Diagram použití programu TestAut.

3.1.5 Popis XML souboru

V souladu s původními požadavky jsou testy i parametry testů ukládány do XML. Jedná se o jeden soubor, který kromě samotných testů uchovává informace o konfiguraci breadboardu a nastavení programu. Některé informace mohou být v XML uloženy na dvou místech v závislosti na tom, zda se jedná o informaci specifickou pro breadboard, který je s použitím daného konkrétního XML testován (pak je uložena v uzlu Definice hardwaru) nebo zda je společná pro všechny breadboardy (pak je uložena v uzlech o úroveň výše). Konkrétně XML obsahuje tyto položky:

Dialogy (DIALOGS) - Definice dialogů otevíraných z menu jako samostatná okna pomocí Windows API. Slouží například k ovládání přepínačů na breadboardu a konfiguraci přístrojů v XML.

Konstanty (CONSTANTS) - Patří sem často používané hodnoty napříč různými testy. Nejčastěji se jedná o určení časového intevralu pro nastavení čekání na různé události.

Sběrnice (LINES) - Definice sběrnic. Obvykle prázdné, využívá se až lokální definice pro konkrétní breadboard.

Zařízení (DEVICES) - Definice zařízení. Jsou zde zařízení, která se se stejným nastavením nacházejí na všech breadboardech.

Parametry (PARAMETERS) - Společné pro všechny breadborady. Obvykle se nevyužívá na této globální úrovni.

Definice hardwaru (HARDWARE_DEFS) - Specifické nastavení daného breadboardu. Právě zde je uložena většina nastavení konstant, sběrnic, zařízení a parametrů.

- **Konstanty (CONSTANTS)** - Hodnoty využívané v testech. Například cesty k důležitým souborům.
- **Sběrnice (LINES)** - Definice všech sběrnic, které se na daném breadboardu nacházejí.
- **Zařízení (DEVICES)** - Definice jednotlivých zařízení připojených k breadboardu. Jedná se o informace o modulech a dalších zařízeních připojených k breadboardu, jako například zdroj elektrického napětí nebo elektrická zátěž.
- **Řídicí jednotky (ECUS)** - Seznam parametrů všech jednotek, které se na breadboardu mohou testovat. Obsahuje informace jako identifikátor jednotky, identifikátory sběrnic, ke kterým je připojena, číslo verze hardware a software a tak dále. Společně s informacemi o sběrnicích a zařízeních dovolují programu správně komunikovat s hardwarem.
- **Parametry (PARAMETERS)** - Hodnoty specifické pro daný breadboard. Především jsou zde uloženy chyby, které na tomto breadboardu vznikají z důvodu nedokonalého nasimulování skutečného vozu, a které je tedy možné při testování ignorovat.

Definice chybových stavů (ERROR_LIST_DEF) - Pojmenování chyb a určení jejich úrovně; cesta k obrázkům, které se používají pro znázornění úspěšnosti testu.

Definice testů (TESTS_DEFINE) - Test je definován buď přímo pomocí parametrů a skriptů nebo odkazem na jiné testy. Hiearchie testů je popsána v podkapitole 3.1.7.

Styl generátoru HTML (GENERATOR_FUNCTIONS) - Parametry pro skript generující výsledky testů udávající grafický styl.

Měření (MEASURING) - Sem se ukládájí hodnoty naměřené během testů. Více o procesu měření je popsáno v podkapitole 3.1.9.

Textové definice (TEXTS_DEFINE) - Textové definice slouží pro uložení textového řetězce přístupného pod zvoleným identifikátorem. Umožňují psát testy a funkce obecnějším způsobem, kdy namísto vkládání konkrétních textových řetězců je na ně pouze odkazováno pomocí jejich identifikátorů. Při potřebě změny těchto řetězců pak není nutné procházet všechny jejich výskyty v testech a funkcích, ale stačí je upravit v dialogu Správce textových definic. Navíc může být pro každou Textovou definici uloženo více textových řetězců a to v jednotlivých jazykových mutacích, přičemž se vždy použije ta, v jejímž jazyce je TestAut spuštěn.

Seznam nápověd (HELP_LIST_DEFINE) - K testům či jejich parametrům může být přiřazena nápověda. Zde jsou nápovědy uloženy ve formě prostého textu, zdrojového kódu v HTML či formou odkazu na soubor.

Funkce (FUNCTIONS) - Obsahuje funkce, které jsou často využívány v různých testech napříč breadboardy - například kontrola správnosti formátu zapsané chyby nebo inicializační funkce.

Uložené testsekvence (SAVED_TESTSEQUENCES) - Jsou zde uloženy často využívané testekvence pro snadnější spouštění.

Dočasné položky (TEMPORARY) - Pro uložení dat ze serveru po spuštění testu, aby byl TestAut po dobu běhu testů nezávislý na připojení k serveru.

3.1.6 Struktura testů

Jednotlivé testy jsou v prostředí programu TestAut nazývány anglickým slovem "teststep". Ty je možné sdružovat do skupin zvaných anglicky "testsequence". V dalším textu budu pro teststep používat prostý výraz "test"a namísto testsequence budu užívat český ekvivalent "sekvence testů". Sekvence slouží pro lepší přehlednost a vznik logické struktury uložení testů podobně jako adresáře v souborovém systému. Narozdíl od souborového systému je ale důležité pořadí testů v sekvenci, protože při každém spuštění sekvence proběhnou testy ve stejném, předem daném pořadí. Testy lze v rámci sekvence přesouvat. Sekvence testů může obsahovat testy a další sekvence. Vzniklá struktura je uložena v XML uzlu "Definice testů"a obsahuje všechny používané testy. Testy jsou seskupovány na základě příslušnosti k jednotlivým druhům testovaných oblastí. Z těchto testů a sekvencí jsou vytvářeny speciální, takzvané "uložené"testsekvence. V XML je pro ně speciální uzel "saved_testsequence". Narozdíl od běžných sekvencí neslouží k uchovávání definic testů, ale k jejich uložení v pořadí a počtu opakování dle požadovaného testovacího scénáře.

Nyní k definici samotného testu. Test je v TestAutu chápán jako kolekce Tcl skriptů a parametrů. V rámci každého testu lze libovolně nadefinovat tyto skripty:

Startup - Je spouštěn na začátku daného testu, obvykle obsahuje nastavení počátečních podmínek.

Main - Je spouštěn po startup skriptu, provádí samotný test.

Terminate - Je spouštěn po Main skriptu, je určený pro vrácení breadboardu do původního stavu.

After error - Obsahuje obsluhu chyb, ke kterým může dojít v ostatních skriptech. Při výskytu chyby je daný test ukončen a pokračuje se dalším skriptem v sekvenci (pokud nějaký existuje). Tento skript tedy zajišťuje, aby vzniklá chyba neovlivnila výsledky následujících testů.

Result - Je spouštěn při generování výsledku testu.

Kromě skriptů může test obsahovat parametry těchto typů:

Číselná hodnota - Číslo s plovoucí řádovou čárkou, ke kterému lze doplnit jednotku a minimální a maximální hodnotu.

Textová hodnota - Prostý řetězec znaků.

Cesta k souboru - Řetězec znaků interpretovaný jako adresa souboru v souborovém systému.

Reference - Odkazuje například na konstanty definované v XML.

Časová hodnota - Pětice celých čísel vyjadřující počet dnů, hodin, minut, sekund a milisekund.

Výběr - Volba jedné hodnoty z definovaných možností.

Výčet hodnot - Seznam textových hodnot oddělených mezerou.

Omezený výčet hodnot - Volba jedné či více textových hodnot z definovaných možností.

Zpráva CAN - Číselný identifikátor a 0 až 8 datových bytů.

Zpráva CAN s maskou - Číselný identifikátor, 8 datových bytů a 8 bytů masky.

3D tabulka - Sada tabulek. Volitelně může platit omezení, které vynutí, aby všechny tabulky měly stejné rozměry (počet řádků a počet sloupců).

Reference na parametr - Může odkazovat na globální Parametry nebo lokální Parametry popsané v XML.

Ke každému parametru mohou být přiřazeny tyto funkce:

Vstupní funkce - Je spouštěna při otevření okna editace parametru.

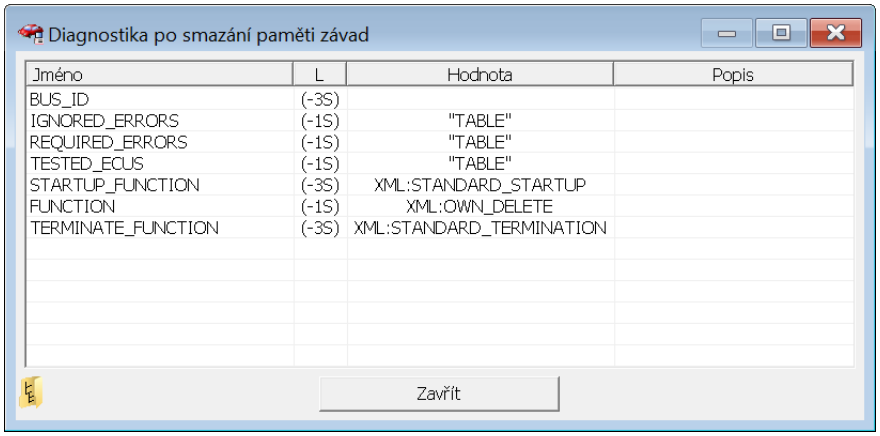
Validační funkce - Slouží pro kontrolu, zda jsou data uložená v parametru validní - tedy zda splňují případné omezující podmínky. Je spouštěna před uložením změn v parametru a lze ji vyvolat ručně v okně editace parametru.

Uživatelská funkce - Lze ji definovat pouze pro některé typy parametrů. Slouží k různým účelům podle uživatelské potřeby. Je spouštěna ručně v okně editace parametru.

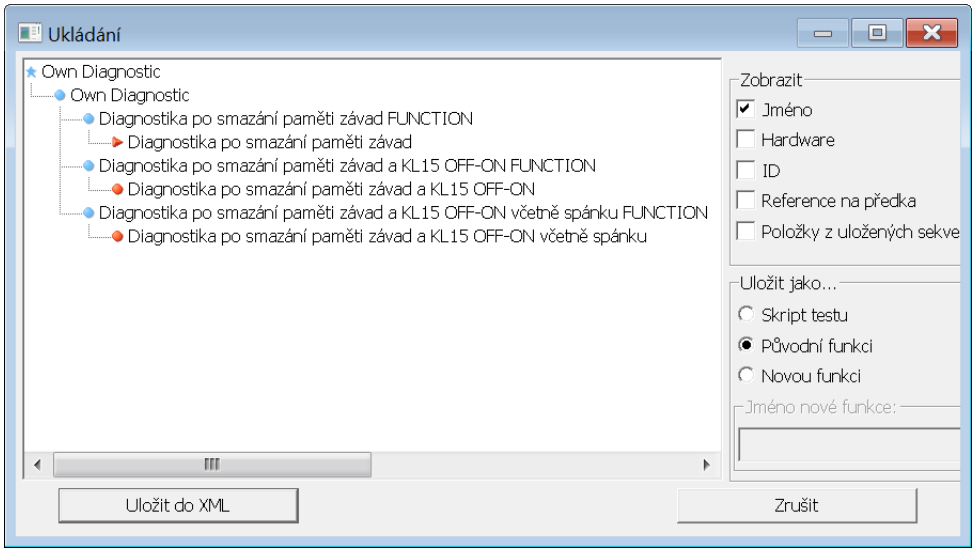
Na obrázku 3.3 je zobrazeno dialogové okno, ve kterém lze prohlížet/upravovat test. Jsou zde vypsané parametry i skripty. U některých je vidět přímo hodnota. Jiné - složitější - lze otevřít v dalším dialogovém okně. Ke každé položce lze připojit krátký komentář. Sloupce S (respektive V a U) indikují přítomnost vstupní funkce (respektive validační a uživatelské funkce). Sloupec L udává o kolik úrovní dědičnosti výše je hodnota definována.

3.1.7 Dědičnost testů

V TestAutu platí pro testy jednoduchá dědičnost. Každý test může mít žádného nebo jednoho předka. Testy se odkazují na své rodiče pomocí jedinečného identifikátoru, který je přiřazen každému testu. Pokud je test bez předka, je nutné všechny parametry a skripty nadefinovat. V opačném případě se test odkazuje na parametry a skripty uložené u jeho rodiče, případně ještě výše v hierarchii dědičnosti. K těmto prvkům odkazovaným k rodičovskému testu lze buď přidávat lokálně definované parametry a skripty a nebo je odkazovat do XML uzlů globální Parametry, lokální Parametry a Funkce. Není ale již možné odkazovat se k parametrům a skriptům definovaným u jiných testů než přímého rodiče - tedy vícenásobné dědění. Jakýkoliv zděděný prvek lze lokálně předefinovat. Při ukládání každého parametru a skriptu je možné zvolit do jaké úrovně dědičnosti aktuální hodnotu uložit - viz obrázek 3.4. Při smazání lokálně předefinovaného prvku se vrátí v platnost zděděná hodnota.



Obrázek 3.3: Ukázka testu.



Obrázek 3.4: Ukázka hierarchie dědičnosti testů.

3.1.8 Skripty

Kromě skrpitů, které jsou uloženy v XML, je využíváno velké množství Tcl zdrojového kódu uloženého ve formě *.tcl souborů v podadresáři "Scripts". Slouží především jako vrstva pro komunikaci s hardwarem, práci se zprávami na CANu, řízení modulů a jednotek, ale používá se také například jako úložiště některých rozsáhlých testů namísto jejich ukládání do XML. Pro organizaci a přehled těchto souborů je součástí TestAutu takzvaný "Správce skriptů". Ten načítá skripty a zobrazuje informace z jejich hlaviček umístěných na začátku každého souboru. Hlavním prvkem hlavičky je číslo verze skriptu. Ta umožňuje jednoduchou správu verzí skriptů. V základním nastavení jsou na lokální počítač zkopírovány skripty ze serveru. V případě potřeby je ale možné skript lokálně předefinovat. Správce skriptů to porovnáním obsahu souborů detekuje a zobrazí tuto informaci. Stejně tak pokud by lokální verze skriptu neodpovídala té serverové. V prostředí správce skriptů lze přepsat lokální skript aktuální verzí z serveru. Také lze nastavit, které skripty se při spuštění programu TestAut budou

porovnávat se serverovými protějšky a v případě zjištění vyšší verze ze serveru automaticky aktualizovat. Naopak nahrávání vyšší verze skriptu na server probíhá editací skriptu v prostředí TestAut. Když při ukládání program detekuje, že je skript ukládán s vyšším číslem verze než se nachází na serveru, nabídne přímo uložení na server. Odtud se změna projeví na ostatních klientech při startu lokálních verzí TestAutu a kontrole aktualizací skriptů.

Ukázka hlavičky skriptu:

```
#header start
#name:Device CAN Modul-CAN6Box
#version:1.00
#executive:Kral, Vrana, Jelinek
#header end
```

3.1.9 Proces měření

Poté, co je v pravém podokně programu připravena testekvence, je možné ji spustit stlačením tlačítka "play". Při každém spuštění je v podadresáři "Result"vytvořena složka s názvem dle aktuálního data a času. Do této složky se zkopíruje aktuální XML, do kterého se navíc do uzlu "Měření"vytvoří potomci všech testů právě spuštěné testovací sekvence a do uzlu "Dočasné položky"se nahraje XML ze serveru. Díky tomu nemůže dojít za běhu testu k jeho selhání z důvodu přerušení komunikace se serverem. Hodnoty probíhajících testů jsou průběžně ukládány do tohoto XML, konkrétně do uzlu "Měření". Na konci testování se na základě dat z tohoto uzlu vytvoří požadovaný výsledek například ve formátu HTML. Je výhodné oddělit získávání dat od jejich vyhodnocování. Například po testování trvajícím několik hodin se zjistí, že je potřeba naměřená data interpretovat nějak jinak. Aby nemuselo celé testování probíhat znovu, je možné v TestAutu díky funkci "Generátor výsledků"zopakovat vyhodnocení již naměřených dat uložených v uzlu "Měření". Při tomto procesu se volají "result"skripty jednotlivých testů, které programátor upravil dle nových požadavků a je vygenerován nový HTML výsledek. Kromě XML souboru a HTML výsledku je při každém testování ukládán ještě log textových výstupů spuštěných testů a log TestAut konzole. Při zjištění chyby (ať už chyby testovaných jednotek či chyby skriptu) je tedy k dispozici velké množství informací k ladění této chyby.

3.1.10 Nasazení programu

V příloze je na obrázku A.1 znázorněn diagram nasazení programu TestAut tak, jak je využíván ve společnosti e4t. Na počítači sloužícím jako server je využívána jedna instance programu TestAutServer, což je jednoduchá aplikace, která spouští miniDOM server s takovým nastavením, aby naslouchal na určitém socketu, odkud budou přicházet požadavky klientů. MiniDOM server je XML parser napsaný v jazyce C++. Slouží k vyčítání a úpravě částí XML souboru. Umožňuje tak práci s XML souborem a jeho sdílení mezi počítači. Pozor však na názvosloví. Program nese název "miniDOM server"nezávisle na tom, zda běží na počítači, který je v roli serveru či klienta. Název je odvozen od skutečnosti, že tento program obsluhuje požadavky, které k němu od ostatních programů přicházejí pomocí obousměrných pojmenovaných rour v rámci počítače nebo pomocí TCP/IP připojení od programů na vzdálených počítačích. MiniDOM server spuštěný na serverovém počítači dokáže kromě XML souboru poskytovat klientům i skripty uložené v podadresáři Scripts. Další důležitou komponentou je Tcl konzole. Slouží k posílání Tcl příkazů interpreteru tohoto jazyka, zobrazuje výstupy s drobnými vylepšeními jako obarvování chyb či vyhledávání v textu.

Pro editaci zdrojového kódu skriptů je využita open source komponenta Scintilla [26]. Do skupiny dynamické dialogy patří okna definovaná v XML uzlu "Dialogy"a spouštěná pomocí MFC. Z programu TestAut lze spustit také generátor výsledků a interaktivní HTML editor nápovědných textů. U Tcl konzole jsou znázorněny balíčky, které skripty spouštěné z této konzole využívají. Některé z nich pracují přímo s hardwarem breadboardu.

Počítač v roli serveru jen jeden. Počítačů vystupujících jako klienti tohoto serveru je více - u každého breadboardu jeden. Existuje i možnost, že dva breadboardy sdílejí jeden počítač, ale tato varianta zapojení se již téměř nepoužívá. Velmi užitečná je možnost spustit TestAut vzdáleně. Je k tomu potřeba mít přístup ke vzdálené složce obsahující spouštěcí souboru TestAutu a pak jej lze spustit v podstatě odkudkoliv. Program TestAut poté běží na počítači ze kterého byl spuštěn, ale využívá soubory vzdáleného počítače. Tímto způsobem lze nastavovat parametry programu dle aktuálního hardwaru, přidávat testy na jednotlivé breadboardy a nastavovat jejich parametry a tak dále. Je možné provádět v podstatě vše kromě samotného spuštění testu, které není možné, protože nelze vzdáleně komunikovat s hardwarem breadboardu.

3.2 EXAM

Tato podkapitola obsahuje popis testovacího nástroje EXAM (*EXtended Automation Method*) vyvíjeného společností MicroNova [1]. Použil jsem dva informační zdroje - prvním z nich jsou oficiální webové stránky tohoto nástroje [7], konkrétně pak dokument prezentující informace o produktu [5] a podrobná dokumentace nástroje [6], která je přístupná po registraci. Registrace je zdarma. Dokonce i plná verze nástroje je k dispozici bezplatně. Vztahuje se na ni freeware licence. Základní knihovny jsou distribuovány s open source licencí. Mým druhým informačním zdrojem byla konzultace s kolegou Ing. Jiřím Černovským, který je jeden ze zaměstnanců společnosti e4t, kteří s tímto nástrojem pracují.

Společnost MicroNova označuje EXAM za standard ve skupině Volkswagen v oblasti testování Hardware-in-the-Loop. Dále je také vhodný k průmyslové automatizaci či k testování Software-in-the-Loop. V době psaní této práce vyšla nová verze EXAMu s číslem 4.1. V této práci budu popisovat EXAM verze 3.4, protože tak mohu využít zkušenosti mých kolegů, narozdíl od nové verze, která ještě není v naší společnosti rozšířena. Grafické uživatelské prostředí je vytvořeno na základě platformy Eclipse [4]. Programovacím jazykem použitým pro psaní testů a ovládání připojených zařízení je Python [24]. Pro EXAM 3.4 se jedná konkrétně o Python 2.5 a pro EXAM 4.1 o Python 2.7. Pro Python existuje velké množství rozšiřujících balíčků a poskytuje výhody objektově orientovaného jazyka. Některé z funkcí nástroje EXAM poskytují uživateli možnost definovat chování testu pomocí diagramu, který je velmi blízký standardu UML, ale v některých drobnostech se od něj liší. EXAM používá jako úložiště dat databáze. Jednu pro testy a jednu pro jejich výsledky. Umožňuje pracovat v režimu klient-server, tedy jeden databázový server využívá současně více uživatelů a data jsou ukládána pouze na serveru. To umožňuje efektivní spolupráci většího množství uživatelů. EXAM detekuje situace, při kterých by mohlo dojít ke kolizi - například při otevírání testovacího případu, který už je otevřený a upravovaný jiným uživatelem je zobrazena hláška informující o vzniklé situaci a je umožněn přístup jen jednomu z těchto uživatelů. EXAM nepodporuje verzování testů, vždy je uložena pouze nejnovější varianta. Všechny objekty v EXAMu jsou součástí jedné hierarchie, která je rozdělena na několik úrovní a umožňuje tak udělit každému uživateli právo ovlivňovat obsah pouze do zvolené úrovně. Každý objekt má (kromě dalších definovaných vlastností) možnost uložit

popis (v anglickém a německém jazyce) a log změn. To je výhodné při spolupráci více lidí nad společnými daty.

Významnou vlastností EXAMu je přenositelnost v něm vytvořených testů mezi obdobnými testovacími sestavami. Tím je myšleno, že testovaná zařízení na jedné sestavě se od odpovídajících zařízení na druhé sestavě mohou lišit softwarem, verzí, typem, výrobcem či dokonce druhem, ovšem za předpokladu, že obě umožňují veškerou funkcionalitu, která je testována. Tedy například při testování dvou různých dveřních ŘJ. Každá může být od jiného výrobce a tedy ovládaná syntakticky rozdílnými příkazy, ale obě musejí být schopné například zamknout zámek, otevřít zámek nebo zasunout okna. Nástroj EXAM této schopnosti dosahuje využitím principu známého z objektově orientovaného programování pod názvem "rozhraní". Odděluje tedy hardwarově nezávislý popis průběhu testu od hardwarově specifických příkazů pro komunikaci se zařízeními.

3.2.1 Použití nástroje

Kompletní průběh práce v nástroji EXAM je znázorněn v příloze na diagramu aktivit A.2. Vstupem je zadání požadovaného testu. Administrátor testů rozhodne, zda bude test vytvořen od základu nový, nebo se modifikuje test již existující.

V následujícím kroku odpovědný pracovník vytvoří formální specifikaci testovacího případu v UML. Tato specifikace musí mít dostatečnou úroveň abstrakce, aby byl test použitelný na různých testovacích sestavách.

Programátor následně napíše zdrojové kódy, pomocí kterých se ovládají jednotlivá testovaná zařízení a meřicí přístroje. Implementace probíhá v jazyce Python, nicméně pomocí jeho různých rozšíření je možné využít i mnoho dalších programovacích jazyků. Toto je jediné místo v procesu tvorby testu, kde je nevyhnutelně nutné používat programovací jazyk. V ostatních krocích buď není programování potřebné anebo je nahraditelné modelováním v UML. To umožňuje práci s nástrojem EXAM i pro uživatele bez programovacích schopností. Pro každé zařízení, které se nachází na některé z možných testovacích sestav, je vytvořena "implementační třída"obsahující metody pro práci s tímto zařízením. Každá implementační třída má přiřazeno jedno rozhraní testu, jehož metody implementuje. K jednomu rozhraní testu může být přiřazeno více implementačních tříd - to znamená, že existuje více hardwarových variant jednoho zařízení. Pro každou testovací sestavu je poté vytvořena takzvaná "systémová konfigurace", což je taková množina implementačních tříd, že pro každé rozhraní přiřazuje nanejvýše jednu implementační třídu. Je zapotřebí zvolit správnou implementaci dle použitého hardwaru. V případě, kdy se v testovací sestavě nachází více stejných zařízení, je nutné, aby každá funkce příslušného rozhraní přijímala parametr určující, které z těchto zařízení má být ovládáno. Pokud by ale tato zařízení byla například od různých výrobců a každé tak vyžadovalo jinou implementační třídu, vznikne problém, protože rozhraní nemůže být implementováno více třídami. V takovém případě je nutné vytvořit více různých rozhraní, což ale kazí obecnost jinak elegantního řešení.

V následujícím kroku se pracuje s takzvanou "sadou parametrů". To je množina pojmenovaných proměnných nesoucích hodnotu. Používá se k popisu testovací sestavy. Například může obsahovat informaci, která jednotka je připojena ke kterému spínači ve spínací skříňce. Tato informace je stejná pro všechny testovací případy prováděné na dané sestavě. Aby se nemusely parametry definovat pro každý z těchto testovacích případů zvlášť, a aby stačilo případnou změnu provést na jediném místě, byly parametry odděleny od testovacích případů a seskupeny do sad parametrů. Nyní je potřeba danému testovacímu případu přiřadit správnou sadu parametrů, a to žádnou nebo jednu. Aby bylo možné rozdělit nastavení celé

sestavy do více částí, a i přes to využít všechny takto definované sady parametrů, je možné mezi sadami parametrů zavádět relaci dědičnosti a to i vícenásobnou.

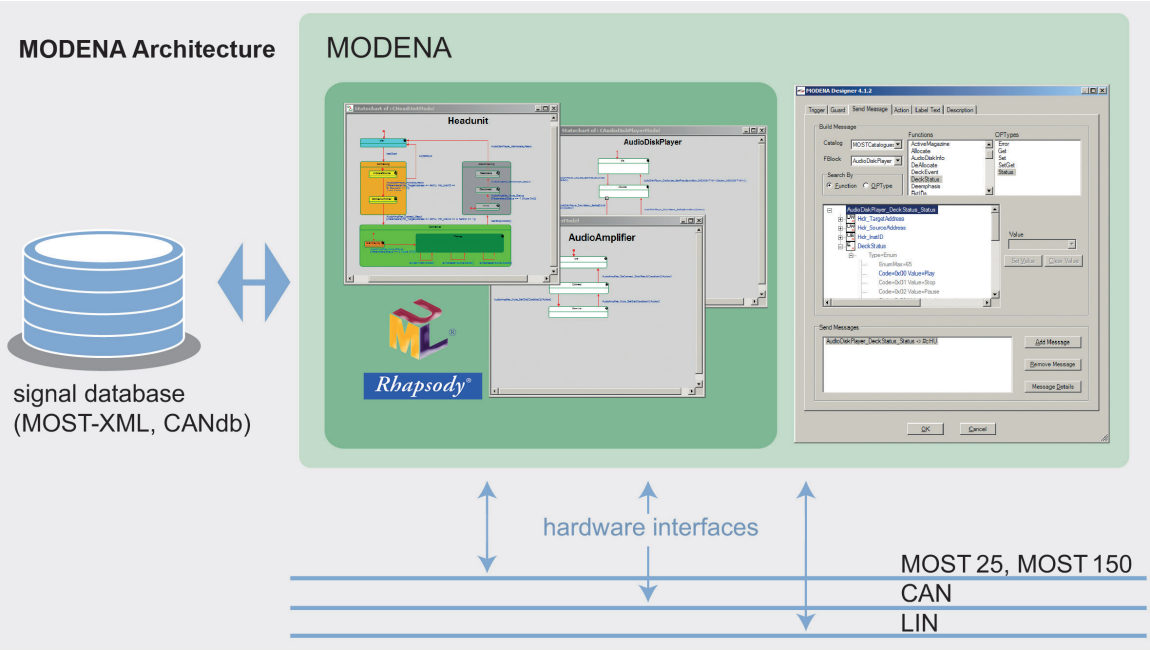
Po přiřazení odpovídající sady parametrů následuje definice průběhu testu. Výsledkem je struktura určující pořadí provádění jednotlivých operací. V tomto místě je možné pro práci využít modelování UML diagramu nebo psaní kódu v programovacím jazyce anebo kombinaci obojího. To se jeví jako nejlepší řešení. Některé triviální operace, jako například výpočty, zapsat ve formě kódu a větší funkční jednotky spojovat formou diagramu. Každý testovací případ se skládá ze čtyř základních částí - inicializace, provádění, deinicializace a reakce na chyby. Některé často prováděné operace - například inicializace zařízení lze uložit jako blok a využívat jej opakovaně v různých testovacích případech. Zápis průběhu testu lze velmi dobře rozdělovat do jednotlivých částí a ty mezi sebou hierarchicky řadit. To je vhodné především pro přehlednost rozsáhlejších projektů.

Tímto je celý test definován a je možné jej pomocí generátoru převést na spustitelný kód. Před samotným spuštěním testu je nutno zadat parametry (například kam se uloží výsledek či jaká bude úroveň logování) a zvolit systémovou konfiguraci dle aktuální testovací sestavy. Průběh testu lze sledovat a tak například odladit chyby při vývoji testu. Po dokončení testu je možné výsledek nejen prohlédnout, ale i komentovat či upravit některá jeho vyhodnocení. Poté může být výsledek vyexportován ve formátu pdf, html a xls.

3.3 MODENA

Informace, které jsem použil pro sepsání této podkapitoly jsem získal jednak z oficiálních webových stránek společnosti Berner & Mattner [11], která je autorem nástroje MODENA a jednak během konzultace s kolegou Ing. Pavlem Náplavou, který tento nástroj používá.

MODENA je systém určený ke specifikaci a testování řídicích jednotek v automobilovém průmyslu a to především ŘJ poskytujících posádce vozu zábavu a informace. Dá se použít ve fázích vývoje ŘJ od specifikace až po integrační testy. Obsahuje knihovny pro snadnou práci se sběrnicemi MOST, CAN a LIN a umožňuje simulovat jejich sítě. Dovoluje simulovat různá selhání ŘJ a je možné provádět offline analýzu. Funkce a testy jsou popisovány na základě modelů. Architektura nástroje je naznačena na obrázku 3.5 převzatého z [12]. Jeho hlavní vlastností je využití IBM Rational Rhapsody [25] pro definici průběhu testů. Ty jsou specifikovány pomocí UML diagramu, což zvyšuje jejich čitelnost, urychluje proces vývoje testu a umožňuje snadnější spolupráci více uživatelů. V některých případech, kdy programátor potřebuje upravit chování tohoto nástroje podle vlastní potřeby, jej taková změna může stát hodně úsilí a různého nastavování. Toto rozhraní poskytuje tvůrci testů možnost vytvářet grafickou cestou stavové stroje, jež jsou vhodným prostředkem pro popis průběhu testu. Jednotlivým částem vzniklého schématu je poté přiřazen zdrojový kód v jazyce C++, který budou provádět. Lze při tom využívat vestavěných knihoven pro práci s hardwarem. Grafický zápis průběhu testu je pomocí IBM Rhapsody taktéž převeden na C++ zdrojový kód. Tyto zdrojové kódy se společně s dalšími pomocnými informacemi ukládají ve formě balíčku - to je souboru s příponou *.sbs. Lze však zvlášť vygenerovat diagramy v různých formátech a zdrojové kody ve formě C++ souborů. Data z MODENA jsou tedy ukládána ve formě souborů. Ty pak mohou být ukládány na server, může na ně být aplikován verzovací systém a tak dále. Výsledkem práce v tomto nástroji je spouštěcí soubor provádějící daný test. Využití architektury CORBA [32] umožňuje testovat hardware připojený k jinému počítači než odkud je test spouštěn, za předpokladu, že jsou oba počítače spojeny počítačovou sítí.



Obrázek 3.5: Architektura nástroje MODENA.

3.3.1 Použití nástroje

Pro testování ŘJ autorádií je v naší společnosti nástroj MODENA použit následujícím způsobem.

Nejsou v něm psány celé testovací případy jako celky, ale pouze jednotlivé kroky, které jsou v testovacích případech využívány. Všechny jednotlivé kroky se tedy specifikují v nástroji MODENA a je vygenerován spouštěcí soubor, pomocí kterého je možné tyto kroky provádět. Testovací případy jsou definovány zvlášť, ve formátu XML. Pro přehlednost a znovupoužitelnost jsou rozděleny do různých souborů a jsou propojeny dědičností. Nastavení testované hardwarové sestavy je definováno v souboru s koncovkou *.ini. Pro spuštění testu je využíván nástroj EXAM a to z důvodu kvalitního a přehledného výsledkového dokumentu, jenž je ukládán do databáze. Test lze ale spustit i samostatně - bez nástroje EXAM, například s využitím webového rozhraní, které MODENA poskytuje. Při spuštění s využitím tohoto nástroje jsou zadány XML soubory specifikující testovací případy, soubor s nastavením hardwaru a spouštěcí soubor vygenerovaný nástrojem MODENA. Spouštěcí soubor se může nacházet v režimu "test mód vypnut"nebo "test mód zapnut". V prvním kroku je spouštěcí soubor aktivován s vypnutým test módem. V tomto případě provede postupně dvě věci. Zaprvé zpracuje požadovaný testovací případ ze všech použitých XML souborů do jednoho XML souboru, pro rychlejší následné zpracování. Zadruhé projde toto XML a načte jej do prostředí EXAM. Poté je spouštěcí soubor aktivován v režimu zapnutého test módu a jsou mu na vstup vkládány názvy a parametry kroků, které má provádět. Spouštěcí soubor provede požadované operace s hardwarem a vrátí výsledek do EXAMu. Ten jej ukládá do databáze. Tímto způsobem je proveden celý testovací případ.

3.4 PROVEtech

V této podkapitole popisuji softwarové řešení PROVEtech. Následující informace jsem získal na webových stránkách společnosti MBtech [8], především v části věnující se nástroji PROVEtech [23], konkrétně pak PROVEtech:TA [2] [22] [20] [17] [18] [19]. Kromě samotných webových stránek byly pro mě zdrojem informací PDF soubory na těchto stránkách prezentované, a to konkrétně prospekt k nástroji PROVEtech:TA [21] a prospekty prezentující novinky nástroje PROVEtech [16] [15] [13] [14].

PROVEtech je komplexní sada nástrojů určená pro použití v oblastech integrace a validace. Byla vyvinuta skupinou MBtech s hlavním sídlem v Německu. Tento produkt je pravidelně aktualizován a rozšiřován. Mezi jeho vlastnosti patří vysoká míra přizpůsobitelnosti zákaznickým požadavkům a přenositelnost mezi operačními systémy. Sada PROVEtech se skládá z těchto jednotlivých nástrojů:

PROVEtech:μHiL - (Hardware-in-the-Loop) Kompaktní a modulární hardwarové zařízení pro funkcionální testování elektronických jednotek během všech vývojových fází a v těsném spojení s vývojovým týmem. Obsahuje také příslušné softwarové vybavení. Je to jedna z variant zařízení, na kterých lze použít ostatní součásti nástroje PROVEtech, a to především PROVEtech:TA a PROVEtech:RE.

PROVEtech:RE - (Runtime Environment) Testovací platforma s dynamicky spustitelnými a konfigurovatelnými komponentami.

PROVEtech:TA - (Test Automation) Automatizované testování jednotlivých řídicích jednotek nebo jejich sítí pomocí vizualizace dat a řízení.

PROVEtech:VA - (Vehicle Application) Flexibilní logování dat z řídicích jednotek, ať už jsou zapojeny v automobilu nebo testovacím stavu.

PROVEtech:RP - (Rapid Prototyping) Integrace a validace komponent softwaru AUTOSAR [33] na počítač, nezávisle na cílovém hardwaru.

Dále je popisován pouze nástroj PROVEtech:TA, protože právě ten je svým použitím nejblíže nástroji TestAut.

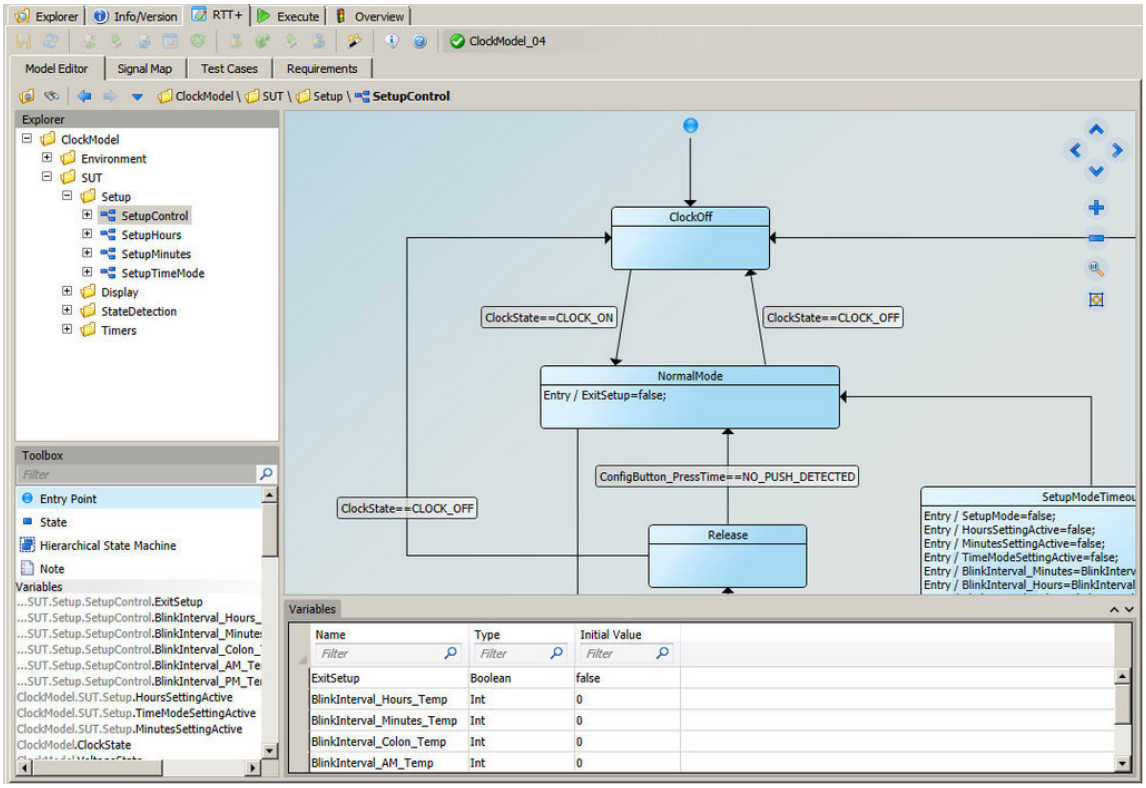
PROVEtech:TA umožňuje uživateli interaktivně nastavovat a měřit všechny stavové proměnné příslušné testovanému systému. Tento nástroj lze použít pro testování ve všech fázích vývoje řídicí jednotky - testování modelu ŘJ (Model-in-the-Loop), testování softwaru ŘJ (Software-in-the-Loop), komponentní testování ŘJ (Hardware-in-the-Loop), integrační testování ŘJ (Hardware-in-the-Loop). Testování lze provádět na testovacím stavu či přímo ve vozidle.

3.4.1 Testy

Testy pro tento nástroj jsou psány v programovacím jazyce Visual Basic s využitím frameworku .NET [10] nebo makro jazyka WinWrap Basic [27]. Lze tedy testy a signály definovat ve formě objektů a využívat výhody objektového programovacího jazyka. Provádění těchto připravených testů může probíhat dvojím způsobem. První možností je klasické provádění spuštěného testu řádek po řádku. Tato varianta je označována jako takzvaný "PC kód", protože se spouští na počítači připojeném k testovací sestavě a postupně dle skriptu vysílá jednotlivé příkazy k hardwaru. Nástoj PROVEtech:TA ale umožňuje ještě jiný přístup

a to takzvaný "real time kód". Tělo testu je před jeho spuštěním přeloženo do speciálního kódu, který je poté nahrán přímo do zařízení na testovacím stavu, které je připojené k řídicím jednotkám. Tímto lze dosáhnout paralelního testování - to je provedení několika akcí v jednom časovém okamžiku oproti spuštění těchto akcí jednotlivými příkazy po sobě volanými. Toto řešení umožňuje lépe nasimulovat některé složité situace, ke kterým může při provozu automobilu dojít.

PROVEtech:TA umožňuje využít volitelné rozšíření "RTT+"pro komunikaci se softwarovým nástrojem společnosti Verified Systems [30]. Toto spojení přináší možnost definovat test pomocí modelu (anglicky Model-based testing). Tvorba testu se pak skládá z vytvoření abstraktní reprezentace průběhu testu pomocí UML diagramu a následném převedení tohoto modelu na zdrojový kód testu, což probíhá z podstatné části automaticky. Jak vypadá práce v prostředí RTT+ lze vidět na obrázku 3.6 převzatém z webových stránek tvůrce [19].



Obrázek 3.6: Ukázka prostředí RTT+ v nástroji PROVEtech:TA.

3.4.2 Práce s daty

Pro uložení testů a jejich výsledků používá PROVEtech:TA databázi; podporuje přitom databázové systémy Oracle, Microsoft SQL Server a PostgreSQL. Využívá tak všech výhod, které relační databáze přinášejí - jako je garantovaná konzistence dat, využití indexování pro urychlení práce s daty a možnost snížení redundance dat vzájemným odkazováním. Pokud je databáze využívána více týmy z různých geograficky vzdálených zemí, je doporučeno použití nástroje SymmetricDS [34], který dokáže replikovat databázi. To znamená vytvoření

více databázových uzlů, kde každý z nich obsahuje všechna data a mezi kterými probíhá asynchronní komunikace pro udržení konzistence dat napříč všemi uzly. Nástroj PROVEtech:TA může být k databázi připojen dvěma způsoby. V takzvaném "Online"režimu jsou připojeny instance programu běžící na počítačích, které jsou připojeny k testovaným zařízením. Pro každý testovací stav může být připojena vždy jen jedna instance programu v tomto režimu. Tato instance má přístup ke všem zařízením a řídicím jednotkám stavu. Druhou možností je "Offline"připojení, které se používá například pro analýzu výsledků testů a vývoj nových testů či úpravu těch stávajících. V Offline režimu lze provádět všechny operace jako v Online režimu, samozřejmě vyjma těch, které využívají testovaný hardware. Pomocí volitelného rozšíření nástroje PROVEtech:TA se lze z Offline instance programu připojit k Online instanci a zadat testy ke spuštění přímo na testovaném hardwaru. Při tomto spuštění nejsou na Offline instanci k dispozici všechny údaje jako na Online počítači, ale je zobrazena informace o tom, v jakém pořadí se testy spustí, zda běží a zda testy již doběhly. Výsledky pak lze plnohodnotně zobrazit z databáze.

Aby mohlo s daty pracovat ve stejnou chvíli co nejvíce příslušných zaměstnanců a přitom si tím navzájem neztěžovali přístup, využívá PROVEtech:TA uživatelských účtů. Každý uživatel se pro využití aplikace musí přihlásit svými přihlašovacími údaji. Každý uživatel má definováno jaké operace může s kterými testy provádět. Tedy zda může test číst, upravovat, spouštět či zobrazovat jeho výsledky. Zdrojové texty testů mohou být pro zvýšení ochrany ukládány do databáze v šifrované podobě. Pokud k jednomu zdrojovému kódu chce přistoupit více oprávněných uživatelů zároveň, systém toto detekuje a uživatele o vzniklé situaci při otevírání textu informuje. Uživatel může zdrojové kódy takzvaně "zamknout"a zajistit tím přístup k takovýmto textům pouze pro sebe. Každý konkrétní test může být v jednom okamžiku spuštěn pouze jedenkrát (na jednom testovacím zařízení). Pokud by z nějakého důvodu bylo vyžadováno spuštění stejného testu paralelně na více počítačích, stačí před spuštěním vytvořit více potomků tohoto testu a ty použít k jednotlivým spuštěním. PROVEtech:TA disponuje vlastní funkcí verzování testů. Při vytvoření nového testu obsahuje řetězec revizí tohoto testu pouze vývojovou verzi testu. Tu je možné plně editovat. Postupně mohou vznikat další vývojové verze a nakonec lze verzi označit za uvolňovací. Tato verze již nemůže být měněna, lze ji pouze číst, odkazovat na ni či ji kopírovat. Pokud je potřeba provést změny na uvolňovací verzi testu, jakoukoliv změnou jejího zdrojového kódu se automaticky vytvoří nová vývojová verze s tímto pozměněným obsahem. Po provedení potřebných úprav lze opět označit tuto verzi za uvolňovací.

3.4.3 Grafické uživatelské prostředí

Okno aplikace se skládá ze dvou hlavních částí. V levé (menší) části okna se nachází takzvaný "kokpit". Je určen pro zobrazení ovládacích a měřicích prvků testovaného systému - v případě automobilu tedy například ukazatel rychlosti, zobrazení polohy přepínačů (ovládání světel, klimatizace, stěračů, a tak podobně), poloha řadicí páky, a tak dále. Všechny tyto prvky lze zobrazit v různých podobách, nejčastěji obrázkem znázorňujícím reálnou podobu prvku a simulujícím jeho reálnou funkci (otáčení, posouvání). Uživatel si tak může vytvořit jakýsi virtuální kokpit, který obsahuje všechny prvky důležité pro aktuální testování, zobrazuje jejich stav a umožnuje jej měnit.

Pravá (větší) část okna obsahuje čtyři podokna, mezi kterými se lze přepínat záložkami. Patří sem "Pracovní plocha", "Manažer testů", "Diagnostika"a "Simulace poruch". Pracovní plocha je hlavním místem, které je používáno při vlastním testování. Stejně jako kokpit slouží pro zobrazení ovládacích a řídicích prvků testovaného systému. Uživatel si

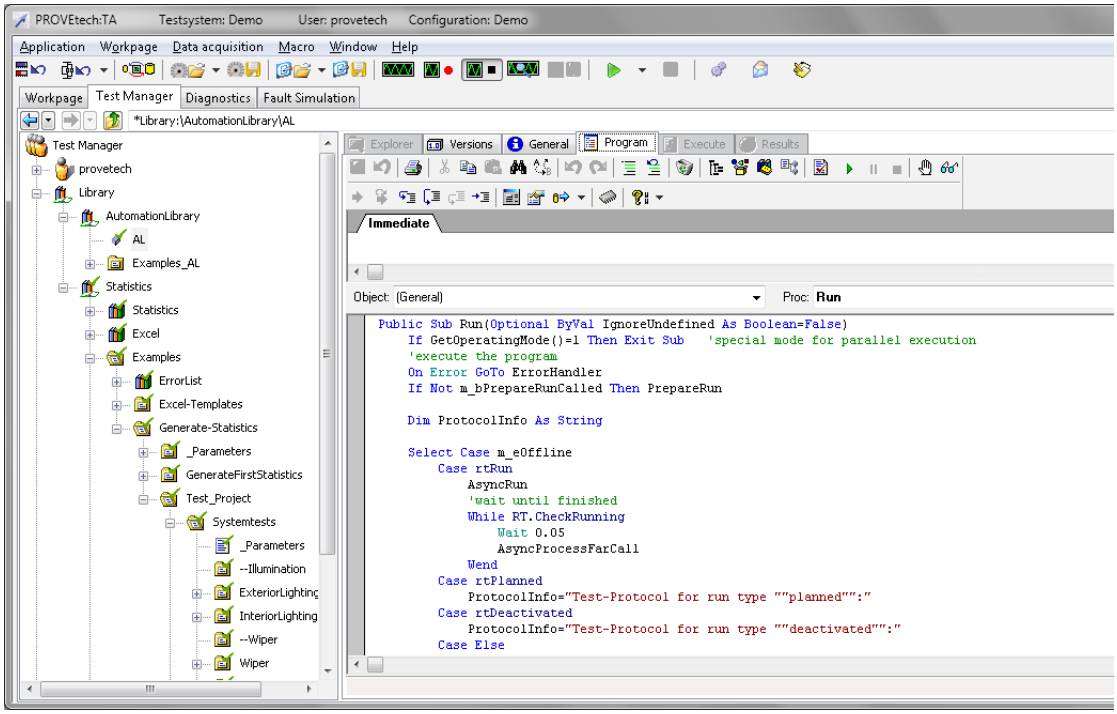
může navolit, které signály zde budou zobrazeny a v jaké podobě. Může zde být zobrazena například rychlost vozu na virtuálním rychloměru, hodnoty proudů měřených na různých místech testovaného systému či hodnoty různých dalších snímačů. Pracovní plocha se může skládat z více takto nadefinovaných obrazovek, mezi kterými je možné se přepínat. Ukázka pracovní plochy je na obrázku 3.7.

Obrázek 3.7: Ukázka pracovní plochy v nástroji PROVEtech:TA [21].

V podokně manažer testů jsou zobrazeny ve stromové struktuře všechny dostupné testy včetně jejich případných verzí, výsledků a protokolů. Procházení stromem nabízí funkce jako například nalezení zdrojového kódu testu, z místa kde je definován pouze odkazem. Nachází se zde i vestavěný textový editor pro úpravu zdrojových kódů testů, jek je vidět na obrázku 3.8.

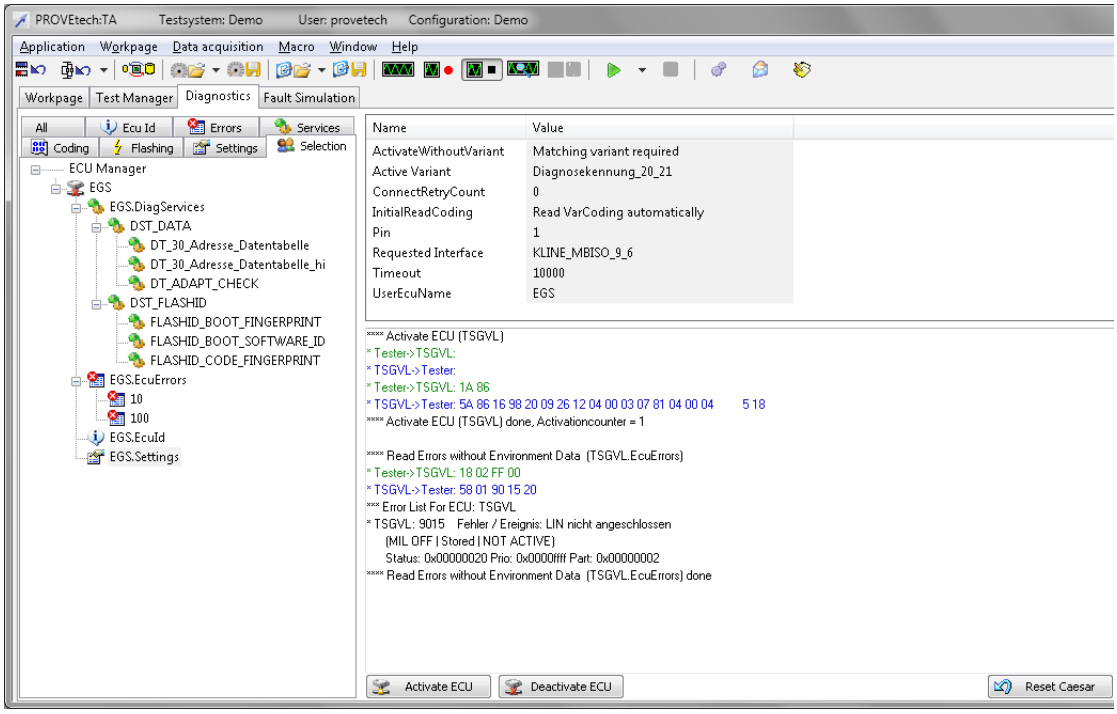
Podokno diagnostika umožňuje přístup ke všem funkcím použitého diagnostického hardwaru. Typicky se jedná o operace vyčítání chyb z ŘJ, její kódování, flashování a zobrazení interních dat. Jak může vypadat podokno diagnostiky je předvedeno na obrázku 3.9.

Simulace poruch je posledním podoknem, které může uživatel zobrazit. Používá se k přerušování elektrického proudu na pinech řídicích jednotek a ke vzájemnému zkratování těchto pinů, po čemž obvykle následuje vyčítání diagnostických informací, zda na vzniklou situaci systém zareagoval dle očekávání. Podokno simulace poruch je uvedeno na obrázku 3.10.

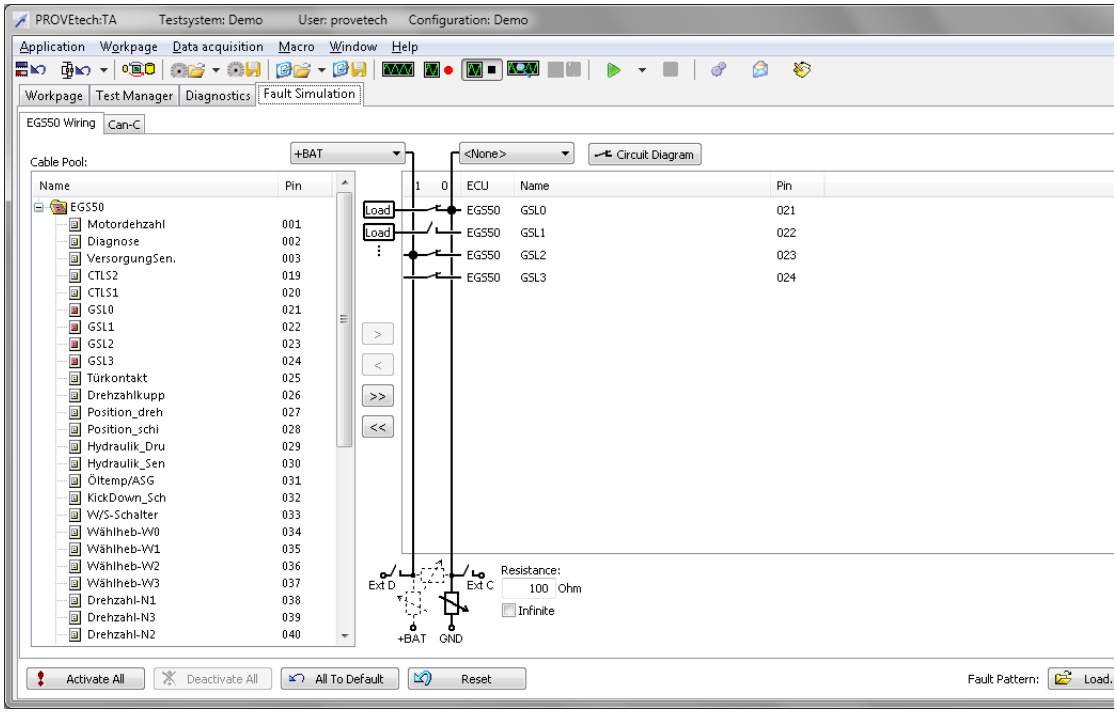


Obrázek 3.8: Ukázka manažeru testů v nástroji PROVEtech:TA [21].

Poslední částí grafického uživatelského prostředí je lišta menu a lišta s ikonami. Slouží k nastavování programu co se týče databáze, práce se signály (přidávání na pracovní plochu, spouštění či pozastavování měření a tak dále), zobrazení oken programu a další funkce potřebné při využívání programu.



Obrázek 3.9: Ukázka diagnostiky v nástroji PROVEtech:TA [21].



Obrázek 3.10: Ukázka simulace poruch v nástroji PROVEtech:TA [21].

Kapitola 4

Vhodné principy pro nasazení na integrační testy řídicích jednotek

V předchozí kapitole byly popsány vybrané programové nástroje umožňující (kromě dalších funkcí) automatizaci integračního testování řídicích jednotek automobilu. Na základě těchto informací následuje v této kapitole souhrn principů, které se mohou jevit jako vhodné pro použití při integračním testování ŘJ ve společnosti Škoda Auto a.s. Jednotlivé principy jsou zde diskutovány a je vyhodnocen jejich přínos. Některé jsou již součástí funkcionality nástroje TestAut, jiné zatím ne. Jako inspiraci pro výběr správných principů jsem použil článek z knihy [37].

4.1 Architektura klient-server

Jedním ze základních konceptů, který obsahují všechny popisované nástroje (včetně nástroje TestAut) je možnost spojit jednotlivé počítače, ke kterým je připojený testovaný hardware, do společné počítačové sítě. Tímto způsobem je možné sdílet data - především testy nebo jejich části. Jeden test tak může být používán na více počítačích a v případě potřeby úpravy testu se změna může snadno distribuovat. Stejným způsobem se můžou šířit nově vytvořené testy. Nástroje samozřejmě fungují i bez komunikace s ostatními počítači, ale všechna data musí být uložena lokálně a změny v testech se projevují jen obtížně (například přenosem do externí paměti). Výše popsané nástroje používají propojení počítačů typu klient-server. Tato varianta přináší centrální úložiště, které spravuje sdílená data. Typicky je to počítač speciálně vyhrazený pro tyto účely a není k němu připojen žádný testovaný hardware. Je vhodné provádět pravidelné zálohování a nasadit i další ochranu takto uložených dat, protože ztráta či nežádoucí modifikace na jednom místě znehodnotí data pro všechny počítače. Pokud je takováto ochrana zajištěna, jedná se o efektivní způsob sdílení dat.

TestAut sice využívá uložení testů na serveru, ale pouze těch, které jsou společné pro více breadboardů. Odtud jsou testy děděny a případně přizpůsobeny konkrétnímu testovanému hardwaru. Tyto podděděné verze jsou však uloženy v XML na lokálním počítači. Žádný z počítačů (včetně serveru) tedy nemá obraz celé hierarchie dědičnosti. Toto je jedna z věcí, které by bylo vhodné upravit a ukládat skutečně všechy testy i jejich potomky na server a ne do lokálních XML. Toto umožňuje kompletní správu testů, a to z kteréhokoliv z připojených počítačů, což je velkou výhodou. V současné době totiž, pokud správce testů potřebuje postupně upravovat lokální testy několika různých breadboardů, musí postupně po jednom spouštět instance TestAutu z jednotlivých počítačů. Toto přepínání je časově náročné a

proto by možnost zobrazení všech testů napříč všemi breadboardy ušetřilo správci testů mnoho času. Proto bude tento princip jedním z úkolů této práce.

4.2 Online a Offline mód

Jednou z věcí, které TestAut nemá a například PROVEtech má, je možnost Offline módu. TestAut totiž automaticky očekává, že k počítači, na kterém je spuštěn, je připojen nějaký testovaný hardware. Je sice možné nástroj používat i na počítačích bez testovaného hardwaru, ale při spouštění hlásí TestAut řadu chyb spojenou s nenalezením očekávaného hardwaru. Proto by bylo vhodné upravit TestAut tak, aby jeho spuštění na počítači, který není určen k testování, ale pouze k tvorbě testů, nevyhledával hardware a bezchybně se spustil. V tomto módu by nebylo možné spouštět testy. Dále by byla vhodná možnost přepínat se mezi Online a Offline módem. Toto bude další z úkolů této práce.

4.3 Vzdálené spuštění testu

Velkou výhodou je možnost spouštět testy z počítače, ke kterému není připojen testovaný hardware, ale který je k počítači s hardwarem připojen počítačovou sítí. To je velmi výhodné pro snadnou obsluhu geograficky vzdálených testovacích sestav. A to jak v rámci budouvy, tak například i mezi více pobočkami společnosti. Nástroj PROVEtech toto umožňuje. Neposkytuje při běhu testu plnou vizuální kontrolu jako při lokálním spuštění, ale pro "PC kód"umožňuje sledovat přibývající výsledky v HTML protokolu. Nástroj MODENA toto řeší možností ovládat vzdálený hardware pomocí architektury CORBA. Nástroj TestAut něco podobného sám o sobě neumožňuje, ale pro tyto účely je ve společnosti Škoda Auto využívána funkce operačního systému Windows Vzdálená plocha. Toto elegantní řešení umožňuje úplnou kontrolu nad vzdálenou instancí testovacího nástroje, včetně celého operačního systému, na kterém je spuštěn. Uživatel tak může vidět průběh celého testu a pracovat s nástrojem úplně steným způsobem jako při práci na lokálním počítači. Navíc je možné spustit více těchto instancí zároveň.

4.4 Použití databáze pro uložení testů

PROVEtech i EXAM používají pro uložení testů relační databázi. Toto řešení umožňuje ukládat velké množství testů a přitom umožňovat rychlý přístup díky indexaci. Při použití v režimu klient-server řeší databáze sama vícesobný přístup a případné konflikty. Je to robusní řešení připravené ukládat rozsáhlé projekty a poskytovat k nim přístup pomocí SQL dotazů.

TestAut z důvodů počátečních požadavků společnosti Škoda Auto využívá od počátku jiné řešení a to XML soubor. Pro jeho zpracování byla vytvořena vlastní implementace funkcí nad DOM modelem. Pro aktuální rozsah testů je toto řešení plně dostačující a i s výhledem do blízké budoucnosti by tato varianta měla stále splňovat všechny kladené nároky, tak jako doposud. Výhodou tohoto řešení je, že veškeré definice testů i nastavení hardwaru se nacházejí v jednom souboru, a tak umožňují jeho snadnou zálohu a případný přesun. Mimo XML jsou uloženy knihovny skriptů, které nejsou přímo testy, ale obsahují funkce pro práci s hardwarem a další pomocné funkce. Distribuce a verzování těchto souborů je řešeno ve Správci souborů, jenž je součástí nástroje TestAut. Také výsledky jsou uloženy samostatně a to v podadresáři Results ve složce TestAut, ze které byl test spuštěn. Každý

spuštěný test je zde uložen ve vlastní složce. Navíc jsou tyto výsledky automatizovaně pravidelně procházeny a ukládá se o nich záznam do databáze. Díky této databázi je pak možné v aplikaci TestAutECUBrowser zobrazovat informace o tom, která řídicí jednotka byla kdy a na kterém breadboardu testována. To jsou důležité informace pro společnost Škoda Auto.

V současné době je velikost XML souboru přibližně kolem 2,5 MB. Takový objem dat je bezproblémově zpracovatelný. XML soubor, který je součástí výsledku (tedy generuje se při běhu testu na základě XML daného počítače a navíc se do něj ukládájí naměřené hodnoty následně použité pro vytvoření výsledku testu), může v závislosti na počtu obsahujících testů nabývat až velikosti kolem 35 MB pro největší používanou sadu testů. I takto velký soubor nevykazuje žádné problémy při jeho zpracování. Proto není žádný důvod na tomto principu ukládání dat něco měnit.

4.5 Tvorba testů formou UML diagramů

PROVEtech, EXAM i MODENA poskytují tvůrci testů možnost definovat test formou diagramu. Do jaké míry uživatel této možnosti využije, je na něm. Tato funkce má dle slov autorů testovacích nástrojů urychlovat tvorbu testů a zvyšovat pochopitelnost testu mezi členy týmu, a to včetně kolegů bez hlubší znalosti programování. Z konzultací, které jsem vedl s mými kolegy, a kteří tyto nástroje využívají, jsem však zjistil, že jim tento způsob tvorby testů v některých případech způsobuje problémy. Všichni lidé, pracující v naší společnosti na psaní testů jsou programátoři a proto nepotřebují zjednodušovat definici testu diagramem. Navíc tato metoda částečně skrývá funkcionalitu programovacího jazyka do komplikovaného systému nastavení. Je však pravda, že když se uživatel naučí psát testy způsobem vhodným pro částečnou tvorbu formou diagramu, může z toho ve většině případů profitovat snadnější prací a tedy i rychlejším vznikem testu. Protože ale tato možnost kromě výhod přináší uživateli i určitá omezení, a protože zjednodušení pro nezkušené programátory není v naší firmě zapotřebí, nebyla tato funkcionalita shledána jako něco, co by nástroji TestAut přineslo výrazné vylepšení.

Kapitola 5

Návrh postupu začlenění vybraných principů do nástroje TestAut 2

V předchozí kapitole jsou popisovány jednotlivé principy, které se jeví jako vhodné pro nástroje provádějící integrační testy. U každého principu je diskutována jeho vhodnost pro použití v nástroji TestAut 2. Vybrané principy vyhodnocené jako přínosné pro stávající nástroj do něj budou zakomponovány. Tato kapitola popisuje návrh jednotlivých kroků, které by měly vést k dosažení tohoto cíle. Tyto návrhy byly také konzultovány s kolegou, Ing. Janem Brabcem, který je spolu s autorem této práce zodpovědný za vývoj a správu nástroje TestAut.

5.1 Výhodnější využití architektury klient-server

Jak již bylo zmíněno v podkapitole 4.1, lze využít architekturu nástroje TestAut pro správu testů výhodněji a to přesunutím kompletně všech testů na jedno místo - do serverového XML souboru. Při spuštění TestAutu se žádné testy nebudou stahovat do lokálního XML, veškerá práce s nimi bude probíhat na serveru. Až při spuštění testu se všechny potřebné testy zkopírují do lokálního XML uzlu TEMPORARY/TESTS_DEFINE.

Stejně tak budou na server přesunuty i Uložené testsekvence. Každá Uložená testsekvence je odkazem vázána ke konkrétní Hardwarové definici a proto nebude problém umístit Uložené testsekvence ze všech breadboardů do jednoho uzlu v serverovém XML souboru.

Kromě samotných testů je výhodné přesunout na centrální místo serverového XML také ostatní data. Především se jedná o Definice hardwaru, které obsahují informace o zařízeních, řídicích jednotkách, sběrnicích, konstantách a parametrech jednotlivých breadboardů. Vyjma uzlu ECUS (Řídicí jednotky) mají tyto uzly své jmenovce v uzlu ROOT každého lokálního XML souboru. Toto rozdělení přináší dvě možnosti kam lze jednotlivé položky uložit. Zda do uzlu pro konkrétní hardwarovou definici (prvek je pak takzvaně *závislý* či *lokální*) anebo do uzlu přímo pod ROOT a tedy vytvořit položku společnou pro všechny hardwarové definice (pak je prvek označován jako *nezávislý* či *globální*). Přesunem uzlů HARDWARE_DEF na server se všechny závislé prvky stanou dostupné z jakéhokoliv počítače přepnutím příslušného hardwaru. Nezávislé prvky naopak zůstanou uloženy v lokálních XML souborech a budou tedy dostupné pouze při spuštění TestAutu nad příslušným lokálním souborem. Tato nevýhoda ale není nikterak velká, protože nezávislých prvků je oproti

těm závislým jen velmi malý počet. Navíc se s nezávislými prvky manipuluje jen velmi zřídka kdy. Pokud by ale přeci jen bylo nutné nezávislý prvek upravovat, mazat či přidávat, lze tak učinit po spuštění konkrétní instance TestAutu nad daným XML souborem. Další možností je zpřístupnit nezávislý prvek na serveru, a to jeho přesunutím (či rozkopírováním v případě více lokálních definic hardwaru) do uzlů závislých. Při spuštění TestAutu bude nabídnuto, který hardware načíst. Definice vybraného hardwaru se na serveru označí příznakem, že je právě editována, aby nebyla upravována z více instancí TestAutu zároveň. Na základě volby hardwaru budou zobrazeny Testy a připraveny Uložené testsekvence vázané k tomuto konkrétnímu hardwaru a také všechny závislé položky uložené v daném uzlu HARDWARE_DEF. Volbu Hardware definice bude možné v rámci běhu TestAutu měnit. Speciální volbou bude varianta ALL, která sice nezpřístupní žádné položky Hardwarové definice, ale umožní spravovat kompletně všechny definované Testy.

Textové definice jsou s úspěchem hojně využívány především pro texty dialogů, které se v průběhu testů mohou zobrazit uživateli. Tyto hlášky jsou zpravidla stejné, ať už se zobrazují na libovolném breadboardu. Proto mohou být uloženy na serveru. Dalším účelem Textových definic bylo pojmenování jednotlivých řídicích jednotek. K tomuto účelu se používaly lokální Textové definice, aby se pojmenování jednotek mohlo lišit pro jednotlivé breadboardy. K tomuto účelu se však již nepoužívají. Protože nebylo nalezeno ani žádné další využití lokálních Textových definic, budou všechny Textové definice přesunuty na server. (Pro uložení textů závislých na aktuálním breadboardu, či dokonce konkrétním hardwaru se využívají takzvané Parametry.) Serverové Textové definice se při startu TestAutu zkopírují do lokálního XML, do uzlu TEMPORARY/TEXTS_DEFINE.

Nápovědné texty lze definovat jak pro testy, tak pro jejich jednotlivé parametry. Testy (ani jejich parametry) obecně nemění svůj význam v závislosti na breadboardu, pro který jsou poděděny. A proto nedochází k situaci, že by stejný Nápovědný text měl mít různý obsah v závislosti na breadboardu, na kterém je definován. Proto budou všechny lokální Nápovědné texty přesunuty na server a při spuštění TestAutu budou zkopírovány do lokálního XML, do uzlu TEMPORARY/HELP_LIST_DEFINE. Pokud by se přesto objevil test (nebo parametr), který by vyžadoval různý obsah Nápovědného textu v závislosti na aktuálním breadboardu, lze jednoduše na každém breadboardu odkázat test (nebo parametr) na jiný Nápovědný text.

Naopak některé položky lokálního XML souboru nejsou vhodnými kandidáty pro přesun na server. Jedná se především o uzly, které jsou měněny jen výjimečně a po většinu doby zůstávají beze změn. Není proto potřeba je centralizovat na server pro jejich snadnější dosažitelnost. To platí například pro uzel GENERATOR_FUNCTIONS, který obsahuje speciální druh testu určený pro generování výsledného protokolu, jenž se již několik let používá stále jeden a ten samý. Dalšími uzly, jejichž obsah se již několik let neměnil jsou uzly DIALOGS, ERROR_LIST_DEF a FUNCTIONS. Uzel FUNCTIONS navíc není na žádném aktuálním breadboradu vůbec využíván (jeho obsah je prázdný). Uzly GENERATOR_FUNCTIONS a FUNCTIONS existují také na serveru a uživatel má proto volbu, zda jejich obsah uloží buď do XML souboru lokálního anebo do XML souboru serverového, čímž zajistí dostupnost daného prvku ze všech připojených instancí nástroje TestAut.

A poslední dva uzly, které v této podkapitole ještě nebyly zmíněny jsou MEASURING a TEMPORARY, které ale z principu patří právě a pouze do lokálního XML.

5.2 Zavedení Online a Offline módu

Pro vyšší komfort uživatelů nástroje TestAut, kteří jej nespouštějí za účelem provádění testů, ale například za účelem správy testů, jejich parametrů či hardwaru, bude přidána možnost spuštění v takzvaném Offline módu. Jak bylo zmíněno v podkapitole 4.2, Offline mód vynechá práci s připojeným hardwarem, takže přeskočí vyhledávání připojených zařízení a nedovolí spouštění testů. Při standardním spuštění bude načten Online mód. Spuštění v Offline módu bude iniciováno parametrem předaným spouštěcímu souboru. Mezi oběma módy bude také možné přepínat. Tato možnost bude přístupná pomocí tlačítka v dialogu Konfigurace. Druhou možností, jak mód přepnout bude pomocí funkční klávesy. Offline mód nebude nic měnit na tom, že definice hardwaru může být vždy otevřena pouze v jedné instanci TestAutu. Aktuální mód bude možné rozlišit příznakem v titulku hlavního okna aplikace.

Kapitola 6

Implementace vybraných vylepšení nástroje TestAut 2

Tato kapitola popisuje průběh implementace vybraných vylepšení nástroje TestAut 2 dle návrhu definovaném v předchozí kapitole (5). Oproti ní jsou ale v této části práce podrobněji rozváděny implementační detaily a popisováno řešení problémů, které během implementace vznikly. Podkapitoly jsou uváděny v takovém pořadí, v jakém byly jednotlivé kroky implementace prováděny. Dříve než bylo možné začít upravovat a přidávat zdrojový kód, bylo nutné analyzovat, jak je stávající zdrojový kód strukturovaný, z jakých souborů se skládá, jaké třídy definuje a jaké objekty spolu jakým způsobem komunikují. Této fázi se věnuje podkapitola 6.1. Poté co byl zdrojový kód analyzován, bylo možné začít s jeho upravováním. Cílem bylo dosáhnout požadovaných změn ohledně přesunu některých dat z lokálního XML souboru do serverového XML souboru, čemuž se věnuje další podkapitola 6.2 a také implementovat Offline mód, což popisuje podkapitola 6.3.

6.1 Analýza stávajícího zdrojového kódu

Vývoj nástroje TestAut probíhá v prostředí Visual Studio 2010. Zdrojové kódy jsou verzovány a sdíleny pomocí nástroje GIT, s využitím prostředí GIT Extensions. Proto bylo pro analýzu využito těchto nástrojů. Řešení Visual Studia s názvem TestAut obsahuje 5 projektů. Kromě hlavního projektu jménem TestAut, obsahuje dále také projekt TestAutServer, což je (jak název napovídá) nástroj, který komunikuje s instancemi nástroje TestAut v roli serveru a spravuje při tom serverový XML soubor. Dalším projektem je ResultGenerator, což je nástroj využívaný v případě, že během testů probíhajících v rámci TestAutu dojde k neočekávané chybě, která zapříčiní, že testovací sekvence nedojde k testu, který generuje výsledný protokol. Naměřená data zůstanou ale uložena v XML souboru a na základě něho vytvoří výslednou zprávu právě ResultGenerátor. Další projekt se jmenuje ActiveXRunExe a jeho výsledek se používá k tomu, aby bylo možné při prohlížení výsledného protokolu v prohlížeči Internet Explorer spouštět pomocí odkazů různé aplikace - například otevřít soubor s logovanými zprávami v příslušném editoru. Posledním projektem TestAut řešení je TestDialog, což je pomocná aplikace pro efektivní testování všech dialogů, které jsou v nástroji TestAut definovány. Kromě tohoto hlavního Visual Studio řešení existují i další, pro jednotlivé samostatné komponenty TestAutu či pro podpůrné aplikace, které se používají ve spojitosti s nástrojem TestAut. Jedná se tak například o aplikaci HtmlEditorForTestAut, která slouží k jednoduchému a uživatelsky přívětivému vytváření a úpravám Nápovědných

textů používaných v TestAutu. Nebo také o komponentu TclConsole, která v TestAutu zajišťuje práci v programovacím jazyce Tcl, či komponentu Tcl_Tedia_20 pro práci s hardwarovým zařízením Tedia. Jednou z nejdůležitějších komponent je však miniDOM, což je vlastní knihovna pro práci s daty ve formátu XML.

Protože hlavní část této práce se zabývá změnami v zacházení s XML daty, je důležité porozumět jaké moduly s těmito daty zachází a odkud a jakým způsobem jsou volány. Jako nejjednodušší způsob jak toto zjistit, se jevilo praktické spouštění TestAutu a TestAutServeru za různých podmínek a pozorování toho, jaké nástroje budou pro práci s XML použity. Správnost zjištěných informací byla následně konzultována s kolegou Ing. Janem Brabcem. Nejběžnějším případem použití je spuštění samotného TestAutu. Ten během své inicializace zjišťuje, zda na stejném počítači již běží instance miniDOM severu. Pokud takovou instanci najde, začne s ní komunikovat. Pokud taková instance zatím neexistuje, TestAut ji spustí. MiniDOM server poskytuje grafické uživatelské rozhraní - takzvaný miniDOM_Monitor, pro zobrazení a procházení všech aktuálně otevřených XML souborů. Každé otevření XML dokumentu (ze souboru či z paměti) se označuje jako *připojení*. Rozlišuje se mezi připojením lokálním a vzdáleným podle toho, jakým způsobem byla přijata žádost o připojení. Lokální připojení je otevíráno na základě žádosti přijaté pomocí obousměrné roury a o vzdálené připojení se jedná, pokud byla žádost přijata skrze TCP/IP spojení. Díky tomuto nástroji bylo zjištěno, že spuštění TestAutu si vyžádalo lokální připojení k těmto pěti dokumentům:

czech.xml - Soubor obsahující textové řetězce pro grafické uživatelské rozhraní dle zvoleného jazyka, v tomto případě v češtině.

run.tmp - Soubor, který je každou vteřinu přepisován ve složce TestAut a obsahuje informace o názvu lokálního počítače a aktuálního uživatele. Slouží pro detekci, zda je příslušný TestAut spuštěn a případně kým.

Scripts.xml - Soubor obsahující informace o použitých skriptech, jejich verzích a časech uložení jejich souborů pro možnost jejich skriptů a detekce lokálních úprav souborů skriptů.

Test.xml - Hlavní XML soubor programu TestAut. Jeho podrobný popis se nachází v podkapitole 3.1.5.

TestAut - main Document - Hlavní dokument programu TestAut. Nachází se v paměti počítače a jeho struktura je stejná jako u XML souboru Test.xml.

Při inicializaci TestAutu je Hlavní dokument naplněn obsahem souboru Test.xml. Během práce v TestAutu se ale obsah Hlavního dokumentu a souboru Test.xml může lišit. Hlavní dokument obsahuje data, která jsou uživateli zobrazována v grafickém uživatelském prostředí. Některá data ale nejsou určena k uložení do souboru Text.xml a jsou ponechána pouze v Hlavním dokumentu v paměti. Jedná se například o identifikační data vyčtená z jednotlivých řídicích jednotek a která se do souboru neukládají, protože se často mezi spuštěními TestAutu mění. Dalším využitím je takzvané *dočasné uložení* používané například pro zdrojové kódy testů, kdy lze vytvářený kód ukládat průběžně do Hlavního souboru a až finální verzi promítnout též do souboru Test.xml.

Dalším případem použití je spuštění TestAutServeru. Stejně jako v případě nástroje TestAut i zde se kontroluje, zda existuje instance miniDOM serveru, kterou si případně vytvoří, a ke které se připojí. Po načtení TestAutServeru je v miniDOM serveru jedním lokálním připojením otevřen soubor TestServer.xml. V případě, že se k počítači, na kterém

je spuštěn TestAutServer, připojí přes síťové rozhraní instance TestAutu, lze zaznamenat, že k miniDOM serveru přibydou dvě vzdálená připojení. Jedná se o připojení od komponenty Tcl konzole a od samotného jádra nástroje TestAut. Obojí využívá soubor TestServer.xml. Připojení má každá z těchto komponent vlastní, aby mohly pracovat vzájemně nezávisle a paralelně. Na straně počítače, ze kterého se instance TestAutu připojuje k serveru je situace taková, že miniDOM server, který zprostředkovává připojení zmíněných dvou komponent ke vzdálenému souboru TestServer.xml, zobrazuje tento soubor jako otevřený s příznakem "SERVER", aby bylo zřejmé, že se jedná o vzdálené připojení směrem odtud. TestAut a TestAutServer mohou dokonce být spuštěny na stejném počítači. Na jejich funkci to nemá vliv. Oba poté sdílejí společný miniDOM server, který při vzdálených připojeních komunikuje "sám se sebou"přes síťové rozhraní počítače.

Dalším krokem v analýze zdrojového kódu bylo využití Debug režimu, který Visual Studio nabízí. V rámci projektu TestAut byl pro každou třídu (vyjma tříd jednotlivých dialogů, kterých je velké množství) umístěn na začátek konstruktoru breakpoint. Taktéž na začátek různých inicializačních funkcí, které byly v kódu nalezeny, byl umístěn breakpoint. Následné spuštění TestAutu v debugovacím režimu dovolilo efektivně krokovat průchod zdrojovým kódem během inicializace a poté i během používání TestAutu. Byly tak zjištěny vazby mezi jednotlivými objekty, odhaleny okamžiky, ve kterých jsou které objekty vytvářeny a byla identifikována "hlavní"třída, která vykonává jádro celého testovacího nástroje. Touto třídou je `CTestAutDlg`. Že bude tato třída velmi důležitá, bylo zřejmé již z toho, že její definice zabírá celý soubor TestAutDlg.cpp, který je zdaleka největší ze zdrojových souborů projektu TestAut. Proto byla pozornost upřena na tuto třídu a byly přidány breakpointy na začátek všech metod, které obsahuje. Tímto postupem bylo vytipováno několik konkrétních metod, které poskytují hlavní funkcionalitu nástroje TestAut. Konkrétně se jedná o metody `BOOL OnInitDialog()`, `BOOL OnCommand(WPARAM, LPARAM)`, `void OnTimer(UINT_PTR)`, `void startTests()`.

Metoda `BOOL OnInitDialog()` je volána v reakci na Windows zprávu `WM_INITDIALOG`, která je dialogu zasílána právě před jeho zobrazením. V této inicializační metodě dochází především k načtení konfigurace ze souboru TestAut.ini, načtení hlavního okna TestAutu a okna Tcl konzole (zatím však nikoliv k jejich zobrazení), připojení k TestAutServeru, načtení lokálního souboru Test.xml, načtení skriptů, synchronizaci Textových definic a Nápovědných textů ze serveru do lokálního XML a načtení připojených zařízení.

`BOOL OnCommand(WPARAM, LPARAM)` je metoda, která je volána knihovnou MFC když uživatel vybere položku v menu, když prvek uživatelského prostředí zašle dialogu zprávu nebo když byla stisknuta klávesová zkratka. Konkrétně obsluhuje 64 funkcí, které lze při práci s TestAutem vyvolat. Jedná se o funkce všech jednotlivých položek menu (jako například spuštění správce skriptů či zobrazení knihovny názvů), reakce na stisknutí všech různých tlačítek (například otevření výsledného protokolu), obsloužení všech poskytovaných gest myší (především práci s testy - jejich přesouvání a kopírování) a klávesových zkratek (například přepnutí mezi jednotlivými okny aplikace).

Pomocí metody `void OnTimer(UINT_PTR)` je obsluhováno 8 různých Windows Timerů, které jsou zde využívány. Slouží k provádění funkcí, které vyžadují pravidelné opakování. Jedná se například o zobrazování aktuálního času a doby běhu testu, o aktualizování částí uživatelského prostředí v závislosti na aktuálním stavu systému, zapisování do stavového řádku aplikace či o zobrazení spořiče obrazovky po daném časovém intervalu.

Poslední ze zmíněných metod je `void startTests()`, která už podle svého názvu slouží ke spuštění testovací sekvence. Proto zajišťuje kontrolu správnosti spouštěných testů, zkopírování potřebných dat ze serveru do lokálního XML, vytvoření nového adresáře pro výsledný

protokol, zkopírování Hlavního XML dokumentu do souboru ve výsledkovém adresáři a jeho příprava pro ukládání naměřených hodnot a spuštění nového vlákna pro testování.

Studiem těchto, ale i dalších metod bylo možné postupně pochopit jednotlivé principy a myšlenky, na základě kterých TestAut funguje. Během tohoto procesu bylo možné narazit na místa, která obsahují známé chyby. Jedná se především o drobné chyby, které nemají vliv na celkovou funkčnost testovacího nástroje. Ale jejich řešení posloužila velmi dobře jako cvičné úkoly pro první pokusy s úpravou stávajícího zdrojového kódu. Jedná se například o opravu dialogu pro uložení výsledného protokolu, který nefungoval ve Windows Vista a novějších Windows operačních systémech, či například o doplnění nedostatečné obsluhy příkazu přepnutí do administrátorského režimu v konfiguračním dialogu.

Po každém vyřešeném nedostatku byly zdrojové kódy uloženy nástrojem GIT pomocí příkazu commit. Taktéž během následujícího vývoje TestAutu byly po každém uceleném balíku úprav zdrojové kódy tímto vyzálohovány a vytvořena tak historie vývoje zdrojových kódů. Procházení této historie směrem k první verzi nástroje bylo dalším zdrojem informací o zdrojových kódech, protože pro každou provedenou změnu bylo jasně viditelné, jakým způsobem byl ovlivněn zdrojový kód.

6.2 Přesun vybraných částí lokálního XML souboru do serverového XML souboru

Přechod na nové uspořádání dat mezi lokálním a serverovým XML souborem spolu s použitím nového spouštěcího souboru nástroje TestAut je dále označován jako *update* tohoto nástroje. Nová verze TestAutu, která je výsledkem této práce, a která již toto nové uspořádání dat pro svou práci vyžaduje, automaticky během inicializační fáze po spuštění kontroluje příznak provedeného updatu v konfiguračním souboru TestAut.ini. Pokud je zjištěno, že update ještě nad tímto lokálním XML souborem neproběhl, jsou při inicializaci TestAutu provedeny veškeré kroky vedoucí k přesunu všech potřebných dat. Update tak probíhá automaticky pro maximální komfort uživatele, který jen volí řešení případných konfliktů, které mohou během tohoto procesu vzniknout. Metoda, která zajišťuje přesun jistých částí lokálního XML na server je volána ve funkci `BOOL OnInitDialog()` právě po načtení vzdáleného a lokálního XML souboru. Následuje popis jednotlivých přesouvaných částí, konkrétního způsobu řešení tohoto přesunu a úprav, které bylo nutné provést, aby TestAut správně pracoval s daty na nových místech a zacházel s nimi podle nových pravidel.

6.2.1 Přesun Textových definic na server

Jako první krok bylo zvoleno přesunutí lokálních Textových definic na server. Toto rozhodnutí nebylo náhodné, ale zakládalo se na odhadu, že tento přesun a následná úprava zacházení s Textovými definicemi by mohly být nejjednoduššími ze všech plánovaných kroků a tedy by bylo vhodné je provést jako první.

Úkolem bylo zrušení lokálního XML uzlu TEXTS_DEFINE společného pro všechny Definice hardwaru daného XML a přesunout jeho obsah do uzlu TEXTS_DEFINE na serveru. Každý uzel TEXTS_DEFINE může obsahovat nula až n uzlů TEXT_DEFINE a každý uzel TEXT_DEFINE může obsahovat nula až n uzlů TEXT, lišících se hodnotou atributu LANG, tedy jazykem, pro který jsou definovány. Během přesouvání uzlu TEXT_DEFINE na server může dojít ke kolizím mezi lokálním a serverovým uzlem TEXT_DEFINE. Kolize nastává v případě, že v lokálním XML i na serveru existují uzly TEXT_DEFINE se stejnou hodnotou atributu ID a zároveň oba dva obsahují alespoň jeden uzel TEXT se

vzájemně stejným atributem LANG. Opakem je nekonfliktní uzel TEXT_DEFINE. Tak je označován nejen lokální uzel, pro který na serveru neexistuje jeho protějšek se stejným atributem ID, ale označuje se tak i lokální uzel, pro který sice existuje na serveru protějšek se stejným atributem ID, ale neexistuje taková hodnota atributu LANG, která by byla definována u některého z lokálních uzlů TEXT a zároveň u některého ze serverových uzlů TEXT. Anebo takové uzly TEXT existují, ale oba dva obsahují stejný text. V případě kolize je nutno rozhodnout, který text má přednost. Toto rozhodnutí je ponecháno na obsluze programu, pomocí dialogového okna, které se zobrazí pro každý lokální uzel TEXT_DEFINE, jenž je v kolizi. Varianty řešení jsou nabídnuty dvě. První možností je upřednostnit lokální definici konfliktního uzlu TEXT_DEFINE a uložit ji na server. Všechny lokální uzly TEXT, které mají v konfliktním serverovém uzlu TEXT_DEFINE své protějšky (uzly TEXT s navzájem shodným atributem LANG), přepíší textovou hodnotu těchto protějšků hodnotou svojí. Všechny lokální uzly TEXT, které nemají v konfliktním serverovém uzlu TEXT_DEFINE žádné protějšky s navzájem shodným atributem LANG, jsou přidány jako potomci konfliktního serverového uzlu TEXT_DEFINE. Pokud serverový uzel TEXT_DEFINE obsahuje kromě toho také nějaké uzly TEXT, které nemají ve zpracovávaném lokálním uzlu TEXT_DEFINE žádné protějšky s navzájem shodným atributem LANG, jsou na serveru ponechány. Druhou možností řešení kolize je ignorování lokálního uzlu TEXT_DEFINE. Pokud ke kolizi mezi lokálním a serverovým uzlem TEXT_DEFINE nedošlo, jsou za serverové TEXT uzly připojeny i lokální TEXT uzly. A konečně v případě, že lokální uzel TEXT_DEFINE žádný serverový protějšek se stejným atributem ID nemá, je tento uzel celý zkopírován jako další potomek uzlu TEXTS_DEFINE na server. Poté, co jsou všechny případné kolize vyřešeny, je ověřeno pomocí atributu VERSION serverového uzlu TEXTS_DEFINE, že po dobu řešení konfliktů nebyl jinou instancí programu TestAut změněn obsah serverového uzlu TEXTS_DEFINE. Pokud byla detekována změna, jsou veškeré připravené uzly k nahrání na server zahozeny a proces vyhodnocení a řešení kolizí je spuštěn od začátku. V případě, že obsah uzlu TEXTS_DEFINE se na serveru po dobu místního zpracování nezměnil, jsou na server nahrány požadované změny v rámci přesunu lokálních textových definic na server a lokální uzel TEXTS_DEFINE je smazán.

Úpravy zdrojového kódu pro práci s Textovými definicemi bylo nutné provést tak, aby nebyl miniDOM dotazován na obsah lokálního uzlu TEXTS_DEFINE, který byl smazán. Namísto něj se nyní využívá uzel TEMPORARY/TEXTS_DEFINE, jehož obsah je naplněn ze serverového uzlu TEXTS_DEFINE při spuštění TestAutu a při spuštění testovací sekvence. V případě, že není Textová definice v tomto uzlu nalezena, je dotaz veden na serverový uzel TEXTS_DEFINE. V grafickém uživatelském prostředí bylo nutné zakázat nabídku úpravy lokálních textových definic a povolit upravovat pouze ty serverové.

6.2.2 Přesun Nápovědných textů na server

Dalším provedeným krokem v rámci úprav zdrojového kódu za účelem dosažení požadovaných cílů této práce bylo přesunutí lokální Nápovědných textů na server. Výběr toho kroku je logický, neboť je do značné míry podobný právě vyřešenému kroku, během kterého byly na server přesouvány lokální Textové definice. Přesouvaný uzel HELP_LIST_DEFINE obsahuje jednotlivé uzly HELP_DEFINE. Každý uzel HELP_DEFINE může obsahovat nula až n uzlů typu TEXT a nula až n uzlů typu FILE. Každý z uzlů TEXT nebo FILE musí mít definován atribut LANG, který určuje, při jakém zvoleném jazyku aplikace bude použit. Uzly TEXT obsahují celé tělo HTML dokumentu tvořícího nápovědu. Uzly FILE obsahují pouze řetězec reprezentující jméno souboru definované jako relativní cesta z adresáře

TestAut2/HtmlHelp. Přesouvání lokálního uzlu HELP_LIST_DEFINE probíhá procesem velmi podobným tomu, kterým se přesouvá uzel TEXTS_DEFINE s několika následujícími rozdíly. Při vyhledávání konfliktů jednotlivých uzlů TEXT (respektive FILE) mezi lokálním a jemu odpovídajícím serverovým uzlem HELP_DEFINE je za konflikt považována i situace, kdy mají odpovídající uzly TEXT (respektive FILE) stejný obsah. Důvodem je to, že se jedná o HTML dokumenty, které mohou obsahovat odkazy na externí soubory, jako například obrázky. Ale i soubor, jenž má v lokálním adresáři HtmlHelp stejné jméno jako soubor existující v serverovém adresáři HtmlHelp, se může od tohoto odpovídajícího souboru lišit svým obsahem. Proto se i v tomto případě, stejně jako v případě ostatních konfliktů, zobrazí kromě dialogového okna upozorňujícího na kolizi, i okno aplikace HTMLEditorForTestAut, v jehož záložkách jsou všechny nápovědy konfliktích uzlů HELP_DEFINE zobrazeny. Uživatel tak může sám porovnat jejich obsah a podle toho se rozhodnout jak konflikt vyřešit. Dalším rozdílem oproti přesouvání uzlu TEXTS_DEFINE je vlastnost, že pro uzly FILE jsou na server zkopírovány i soubory, na něž uzly odkazují. Po přesunu všech uzlů HELP_DEFINE na server již soubory v lokálním adresáři HtmlHelp nejsou potřebné a proto je tato složka smazána. Ještě předtím je její obsah zkomprimován do .zip souboru a uložen do odpovídajícího adresáře backup. Posledním rozdílem oproti přesouvání uzlu TEXTS_DEFINE je to, že detekce situace, zda se během řešení kolizí změnila verze uzlu HELP_LIST_DEFINE na serveru, je prováděna po každém zpracovaném uzlu HELP_DEFINE, aby v případě, že tato situace nastane, nemusel uživatel řešit všechny konflikty znovu.

Porovnání verzí uzlů HELP_LIST_DEFINE se provádí pomocí atributu VERSION, jenž je sice v XML použit, ale obsluha zajišťující jeho využití ve zdrojových kódech chyběla. Proto byla tato obsluha ve všech potřebných místech doplněna. Jednalo se především o zvýšení verze uzlu HELP_LIST_DEFINE při úpravě či smazání uzlu HELP_DEFINE nebo jeho části. A také při uložení otevřeného souboru v aplikaci HTMLEditoForTestaut, což vyžadovalo úpravu zdrojových kódů této aplikace a proto je nutné pro správnou funkci šířit tuto novou verzi aplikace spolu s novou verzí nástroje TestAut. Další úpravou bylo stažení serverového uzlu HELP_LIST_DEFINE do lokálního uzlu TEMPORARY a jeho upřednostňování při načítání HTML nápovědy. Aktuální verze je ze serveru stažena také při každém spuštění testovací sekvence a při jakékoliv úpravě tohoto uzlu prováděné z lokálního TestAutu. Bylo také nutné odebrat podokno v dialogu Správce nápovědných textů, které bylo určeno pro editaci lokálních Nápovědných textů. Další úprava zdrojového kódu řešila chybu při využívání těchto Nápovědných textů, jenž byla již nějakou dobu známa. Na Nápovědné texty mohou odkazovat dva typy entit. A těmi jsou testy a parametry testů. Odkaz je proveden pomocí atributu HELP_REF. V případě, že jsou testy děděny a vzniká tak jejich hierarchie, může mít každý potomek testu vlastní odkaz na Nápovědný text. V případě, že odkaz na Nápovědný text nemá, prochází se postupně jeho rodiči až ke kořenovému testu a je použit první nalezený odkaz na Nápovědný text. A zde byla zmíněná chyba. V případě, že test Nápovědný text pouze dědil od svého předka, byla sice správně zobrazena ikona nápovědy, ale po jejím aktivování se žádný Nápovědný text neotevřel. Toto nesprávné chování bylo opraveno.

6.2.3 Přesun definic testů na server

Na základě zkušeností získaných během přesunu Textových definic a Nápovědných textů bylo možné přistoupit ke komplikovanějšímu procesu, kterým je přesun samotných testů. Přesun jako takový příliš komplikovaný není. Náročná byla následná úprava zdrojových

kódů tak, aby se pracovalo s definicemi testů pouze na serveru. Práce s testy je totiž velmi podstatnou a dá se říci až hlavní složkou funkcionality nástroje TestAut.

Před započetím práce na přesunu testů na server byla provedena analýza stávajícího stavu správy testů. V této podkapitole bude slovo *test* označovat společně Testy (uzly TESTSTEP) a Sekvence testů (uzly TESTSEQUENCE), nebude-li uvedeno jinak. Každý test by měl mít definováno, pro jaký hardware je určen. K tomu slouží uzel HARDWARE_LIST, který je potomkem uzlů TESTSTEP a TESTSEQUENCE. Tento uzel může obsahovat žádný, jeden, nebo více identifikátorů hardwaru. Pokud nemá test definován hardware tímto způsobem, platí pro něj hardware definovaný u jeho rodiče, což platí rekurzivně. To, jaké testy budou v levém podokně uživatelského prostředí zobrazeny, záleží na zvoleném hardwaru ve vysouvací nabídce nad tímto podoknem. Po přepnutí do administrátorského režimu lze navíc kromě identifikátorů jednotlivých hardwarových definic obsažených v lokálním XML souboru zvolit také možnosti *ALL* a *SERVER*. Při volbě ALL, jsou zobrazeny všechny testy uložené v lokálním XML. Naopak při volbě SERVER jsou zobrazeny všechny testy uložené v serverovém XML. Ve vysouvací nabídce volby hardwaru se mohou kromě právě popsaných možností vyskytnout i další řetězce, které nejsou vázané k žádnému konkrétnímu hardwaru a ani nepatří mezi speciální možnosti ALL a SERVER. Jedná se například o řetězce *SHARED* a *UNIVERSAL*. Tyto řetězce totiž byly obsaženy v některých uzlech HARDWARE_LIST, ačkoliv nejsou vázány se žádnou konkrétní definicí hardwaru. Jejich účelem je vyjádření skutečnosti, že jsou tyto testy nezávislé na konkrétním hardwaru a z těchto testů se následně dědí potomci pro jednotlivé hardwarové definice, ve kterých se již vyplňují parametry závislé na konkrétním hardwaru. Po analýze všech XML souborů z jednotlivých breadboardů bylo ovšem zjištěno, že k vyjádření této univerzality testu se v uzlech HARDWARE_LIST používá hned několik různých řetězců a to SHARED, UNIVERSAL, ALL a v serverovém XML i SERVER. Proto bylo přistoupeno ke sjednocení tohoto značení a nově budou možnosti výběru hardwaru (a potažmo testů) tyto: *ALL* - Zobrazí veškeré definované testy, které jsou nyní všechny uložené v serverovém XML v uzlu TESTS_DEFINE. *UNIVERSAL* - Pouze tato možnost bude používána pro označení testů, které jsou definovány obecně a nejsou určeny pro konkrétní hardware. Dalšími možnostmi budou již pouze jména jednotlivých hardwarových definic. Dosáhnout tohoto nového čistého stavu bude vyžadovat úpravu stávajících definic testů. Některé kroky lze automatizovat, jiné bude nutno provést ručně před samotným nasazením nové verze nástroje TestAut, protože vyžadují individuální posouzení, jak se daný test aktuálně používá a jaký by na základě toho měl mít přiřazený hardware dle nového zjednodušeného systému označování. V lokálním XML musí být všechny testy patřící jednomu konkrétnímu hardwaru označeny v uzlu HARDWARE_LIST jeho identifikátorem. Dále musí být všechny testy v lokálním XML, které jsou určeny jako sdílené mezi všemi lokálními hardwarovými definicemi, označeny v uzlu HARDWARE_LIST všemi příslušnými identifikátory hardwaru oddělených mezerou. V serverovém XML je potřeba provést ručně podobné úpravy. U testů určených pouze některým hardwarovým definicím, musí být v uzlu HARDWARE_LIST definovány příslušné identifikátory hardwaru. A u testů obecných, nezávislých na konkrétním hardwaru musí být v uzlu HARDWARE_LIST uveden řetězec UNIVERSAL. Ve většině případů jsou právě popsaná pravidla již dodržena, ale není tomu tak vždy a proto je nutné zajistit jejich dodržení.

Následují úpravy, které je již možné provést strojově jako součást přesunu definic testů na server. V lokálním XML se jedná o nahrazení řetězce ALL v uzlu HARDWARE_LIST identifikátory všech lokálně uložených hardwarových definic. V serverovém XML je nutné nahradit všechny řetězce SERVER obsažené v uzlech HARDWARE_LIST řetězci UNIVER-

SAL. Dalším krokem v přesunu testů na server, je zajištění unikátnosti jejich identifikátorů v rámci celého lokálního i serverového XML. Při konfliktu identifikátorů je přesouvanému testu vygenerován nový, unikátní identifikátor, a je nutné také projevit tuto změnu ve všech referencích na tento test. Reference mezi testy je také nutné upravit tak, aby se používaly pouze jako atribut PARENT_REF a již nikoliv jako atribut SERVER_REF, který značil, že rodičovský test se nachází na serveru a nikoliv v lokálním XML. Posledním krokem je vlastní přenesení připravených definic testů z lokálního uzlu TESTS_DEFINE do serverového uzlu TESTS_DEFINE a jeho smazání v lokálním XML souboru.

Jak již bylo zmíněno v úvodu této podkapitoly, je práce s testy nejpodstatnější složkou nástroje TestAut. Důsledkem toho je, že se zacházení s testy objevuje na mnoha místech zdrojového kódu. Nejpodstatnější, ne však jedinou změnou, kterou bylo potřeba provést, byla změna XML dokumentu, který se při správě testů musí používat. Již to není lokální XML soubor a jeho kopie v paměti počítače, ale nyní je to serverový XML soubor. Proto byla ve zdrojovém kódu zkontrolována všechna místa, kde se pracuje s lokálním XML souborem, s lokálním XML dokumentem v paměti, s názvem lokálního XML souboru, s funkcemi ukládajícími prvky do lokálního XML souboru, s funkcemi ukládajícími prvky do lokálního XML dokumentu v paměti, dále byly také zkontrolovány všechny výskyty řetězců, které označují některé důležité uzly, které byly přesouvány (jako například TESTS_DEFINE, TESTSEQUENCE, TESTSTEP a HARDWARE_LIST). Na těchto a dalších místech, kam z těchto míst vedly odkazy, bylo zajištěno, aby byl využíván správný XML dokument. Ne vždy totiž práce s testy musí probíhat na serveru. Například práce s testy v pravém podokně programu probíhá v lokálním XML dokumentu, konkrétně v uzlu MEASURING. Totéž platí pro práci s testy, které se při startu testovací sekvence zkopírují ze serveru do lokálního XML dokumentu do uzlu TEMPORARY.

Dále byly upraveny podmínky, za kterých lze mazat a přidávat testy. Dříve byl vyžadován administrátorský režim a zvolená možnost ALL nebo SERVER. Nyní postačí být v administrátorském režimu, aby bylo umožněno manipulovat takto s testy i při volbě jednotlivých hardwarových definic.

6.2.4 Přesun uložených sekvencí testů na server

Kromě přesunu definic testů bylo dalším cílem přesunout i sekvence, které jsou z těchto testů poskládané a určené ke spouštění. Stejně jako v případě samotných testů bylo nejprve nutné stejným způsobem zajistit unikátnost identifikátorů v rámci celého serverového i lokálního XML. Další přípravné kroky již nejsou v případě uložených sekvencí testů potřeba a je možné je přesunout z lokálního uzlu SAVED_TESTSEQUENCES do serverového uzlu stejného jména a lokální uzel smazat.

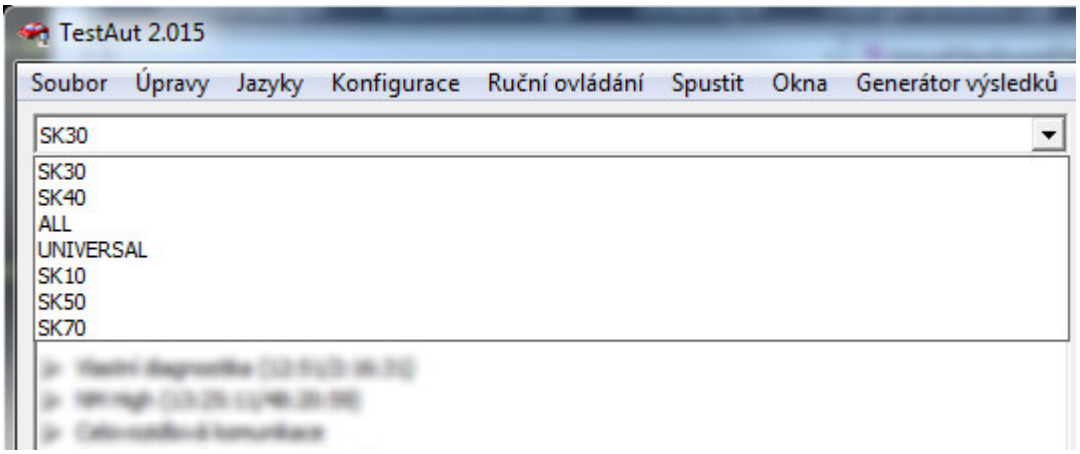
Opět byla vyhledána všechna místa ve zdrojovém kódu, která pracují s uloženými sekvencemi a bylo zajištěno, aby se odkazovala do správného, tedy serverového, XML dokumentu. Oproti definicím testům není nutné stahovat uložené sekvence po spuštění testovací sekvence do lokálního uzlu TEMPORARY, protože za běhu testů již nejsou jejich údaje potřeba. Jsou totiž vždy překopírovány z uzlu SAVED_TESTSEQUENCES již při načtení sekvence a to do uzlu MEASURING.

6.2.5 Přesun hardwarových definic na server

Poslední, ale neméně důležitou částí lokálního XML, která je určena k přesunu na server je uzel HARDWARE_DEFS. Lze jej označit jako druhou nejdůležitější část dat, se kterými TestAut pracuje, hned po testech. Definice hardwaru, respektive obsah tohoto uzlu, je

popsán v podkapitole 3.1.5. Přesun dat na server opět obsahuje zajištění unikátnosti identifikátorů v rámci celého serverového i lokálního XML. V tomto případě je ovšem aplikována pouze na identifikátor uzlu HARDWARE_DEF. Všechny ostatní identifikátory obsažené v potomcích tohoto uzlu, již nemusí být unikátní v rámci celého serverového XML, ale pouze v rámci tohoto uzlu HARDWARE_REF. Důvodem je například situace, kdy více hardwarových definic sdílí nějaký test, včetně jeho parametru, v němž jsou uvedeny řídicí jednotky pomocí svých identifikátorů. Tyto jednotky, ačkoliv patří k různému hardwaru, sdílí stejný identifikátor. Tyto potencionální kolize nezpůsobí žádné chyby, pokud bude zajištěno, že při práci s obsahem uzlu HARDWARE_DEF bude přistupováno vždy do správné hardwarové definice. V případě, že identifikátor uzlu HARDWARE_DEF není unikátní a je mu přidělen identifikátor nový, je tato změna promítnuta také do všech atributů HARDWARE_LIST a ACTIVE_HARDWARE ve všech definicích testů a uložených sekvencí. Oproti tomu jméno hardwaru (to je obsah uzlu NAME, který je přímým potomkem uzlu HARDWARE_DEF) může být mezi několika definicemi hardwaru společné. Jedná se o situace, kdy různé breadboardy představují stejný typ vozidla.

Novým pojmem, který je zaveden s přechodem na novou generaci nástroje TestAut, je *lokální hardware*. Protože bude nyní možné přepínat na hardwarové definice, které nepocházejí lokálního XML souboru a tedy nejsou určeny pro tento breadboard, musí být nějak odlišeny od těch, které naopak byly původně uloženy v lokálním XML. Rozdíl je podstatný. Při přepnutí na lokální hardware lze vyhledávat zařízení a jednotky a lze spouštět testy. Ale pro přepnutí na hardware, který není lokální, toto provádět nelze. Způsob, jakým lze lokální hardware odlišit od ostatního hardwaru, je uložení příslušných identifikátorů do konfiguračního souboru pomocí dialogu Konfigurace. Během přesunu hardwarových definic na server se tato položka vyplní automaticky identifikátory hardwaru, který byl lokálně definován. Aby bylo v administrátorském režimu při výběru hardwaru z vysouvací nabídky zřejmé, který hardware je lokální, jsou zobrazena shora dolů nejprve všechna lokální hardwarová jména, poté speciální řetězce ALL a UNIVERSAL a následně ostatní hardware. Ukázka je na obrázku 6.1, kde jsou SK30 a SK40 jména lokálních hardwarových definic a SK10, SK50 a SK70 jsou jména ostatních hardwarových definic.



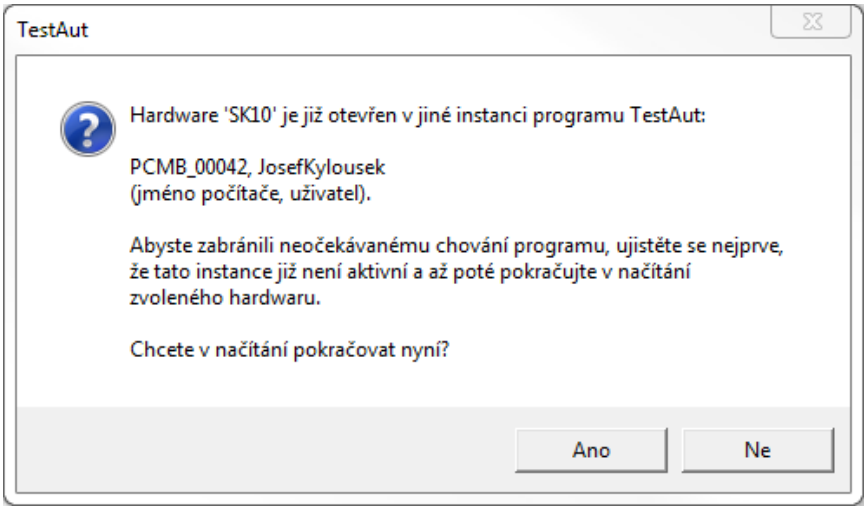
Obrázek 6.1: Ukázka nové vysouvací nabídky pro volbu hardwarové definice.

Poté, co jsou vyřešeny konflikty identifikátorů a vyplněn lokální hardware, lze přejít k přesunu všech lokálních uzlů HARDWARE_DEF do uzlu HARDWARE_DEFS v serverovém XML a jejich smazání v lokálním XML souboru.

Úpravy zdrojového kódu nutné pro správné zacházení s definicemi hardwaru, které se nyní nachází na jiném místě, obnášely obvyklé přepisování dokumentu z lokálního na serverový. Dále byla přidána kompletní obsluha lokálního hardwaru, aby TestAut správně vyhodnotil, co je možné se kterým hardwarem provádět a aby uživatel mohl definovat, který hardware je lokální. Také bylo přidáno zkopírování aktuálně zvoleného uzlu HARDWARE_DEF ze serveru do lokálního uzlu TEMPORARY po spuštění sekvence testů, aby v případě výpadku serveru během testování byly stále dostupné veškeré informace, které jsou v tomto uzlu uloženy. Další podstatnou změnou která byla provedena, bylo přidání atributu OPENED uzlům HARDWARE_DEF. Vždy, když je v TestAutu ve vysouvací nabídce zvolená konkrétní hardwarová definice, je k jejímu uzlu tento atribut přidán a jeho hodnota obsahuje řetězec tvořený názvem počítače a jménem uživatele, dle aktuální situace, za které byl TestAut spuštěn. Po přepnutí na jiný hardware nebo při ukončení TestAutu je tento atribut u daného hardwaru smazán. Tento atribut slouží k tomu, aby bylo zabráněno otevření stejného hardwaru ve více instancích TestAutu. Taková situace by totiž mohla vést k nesprávnému chování TestAutu, protože by jedna instance mohla například mazat data, která jsou aktuálně zpracovávaná instancí druhou. Tímto jsou chráněny definice testů, uložené sekvence testů i všechny prvky uložené přímo v rámci uzlu HARDWARE_DEF. Tato ochrana může být validní cestou obejita a to přepnutím do administrátorského režimu a zvolením hardware volby ALL. Tímto se zpřístupní všechny testy, včetně těch, které patří k jinde otevřenému hardwaru. Případné riziko je eliminováno tím, že v administrátorském režimu můžou pracovat pouze lidé, kteří případné hrozby znají a umí se jim vyhnout. Ostatní prvky, jako uložené sekvence a obsah uzlů HARDWARE_DEF budou i v tomto případě skryté. Protože může nastat situace, kdy se TestAut, pro který je právě otevřená některá definice hardwaru může zaseknout a přestat pracovat, případně může být náhle přerušeno spojení mezi ním a TestAut serverem, existuje možnost otevřít i hardwarovou definici, která má neprázdný obsah atributu OPENED. V případě pokusu o otevření hardwarové definice, která je již otevřena jinou instancí nástroje TestAut (případně se z důvodů popsaných v předchozí větě chybně neuzavřela), zobrazí nástroj TestAut chybovou hláškou, jaká je vidět na obrázku 6.2.

6.3 Rozdělení chodu aplikace na Online a Offline mód

TestAut již obsahuje podobnou funkcionalitu a to detekci, že nástroj není spouštěn z lokálního počítače, ale ze vzdálené (síťové) složky. V tom případě je přeskočeno vyhledávání připojených jednotek. Tato funkcionalita bude pro Offline mód rozšířena o zablokování spouštění testovacích sekvencí a zablokování příkazu spuštění vyhledávání zařízení. Po přepnutí do Online módu budou všechny popisované funkce opět zpřístupněny. Uživateli bude aktuální stav zobrazen tak, že v případě Offline módu přibude do titulku hlavního okna za název programu a jeho verzi ještě řetězec *OFFLINE mód*. V případě Online módu bude v titulku hlavního okna pouze název programu a jeho verze. Přepínání módů bude ošetřeno stejně jako je tomu u administrátorského módu. V dialogu Konfigurace přibude pro tento účel další tlačítko. Jako funkční klávesa, která módy přepne, byla zvolena klávesa F9. TestAut bude spuštěn přímo v Offline módu, pokud bude zjištěn parametr spouštěcího souboru *–offline*.



Obrázek 6.2: Ukázka chybové hlášky zobrazované při pokusu o načtení jinde otevřené hardwarové definice.

Kapitola 7

Testování provedených modifikací

Všechny provedené změny i nově napsané části zdrojového kódu byly pečlivě otestovány a to jak jednotkově - tedy každý ucelený blok samostatně, tak i integračně - tedy nasazením nové verze nástroje TestAut 2 na počítač obsluhující skutečný breadboard a ověřením správné komunikace mezi všemi upravovanými částmi nástroje TestAut. Součástí této kapitoly je také vyhodnocení zpětné vazby od uživatelů nástroje a jsou zde diskutovány dosažené výsledky.

7.1 Jednotkové testy

Následuje popis testování jednotlivých kroků provedených k dosažení požadovaných změn funkcionality nástroje TestAut 2.

7.1.1 Testování přesunu Textových definic na server

Během přesouvání každého lokálního uzlu TEXT_DEFINE na server mohou mezi ním a jeho serverovým protějškem s navzájem stejným atributem ID nastat tyto vztahy jednotlivých uzlů TEXT:

Shoda - Lokální i serverový uzel TEXT mají stejný textový obsah.

Konflikt - Textový obsah lokálního a serverového uzlu TEXT se navzájem liší.

Pouze lokální definice - Lokální uzel TEXT nemá v serverovém uzlu TEXT_DEFINE žádný protějšek se stejným atributem LANG nebo pro lokální uzel TEXT_DEFINE neexistuje žádný protějšek na serveru.

Pouze serverová definice - Serverový uzel TEXT nemá v lokálním uzlu TEXT_DEFINE žádný protějšek se stejným atributem LANG nebo pro serverový uzel TEXT_DEFINE neexistuje žádný protějšek v lokálním XML souboru.

V případě, že řešením konfliktu je zvoleno použití lokálního uzlu TEXT_DEFINE a jeho uložení na server, musí dojít k přepsání konfliktního uzlu TEXT na serveru lokálním uzlem TEXT, všechny lokální TEXT uzly bez serverového protějšku musí být na server nahrány a všechny serverové uzly bez lokálního protějšku musí být na serveru ponechány. V případě že řešením konfliktu je zvoleno ponechání serverové definice, jsou konfliktní serverové TEXT uzly beze změn, stejně tak serverové uzly TEXT bez lokálního protějšku zůstávají ve stejném stavu a lokální uzly TEXT bez serverového protějšku jsou zahozeny.

Protože existují 4 různé situace, které mohou nastat a bylo by vhodné otestovat všechny možné kombinace těchto situací, vzniká 4^2, tedy 16 různých typů uzlů TEXT_DEFINE. Na základě tohoto poznatku byl sestaven lokální uzel TEXT_DEFINE (viz přílohu B.1) a serverový uzel TEXTS_DEFINE (viz přílohu B.2). Tyto uzly byly vloženy do příslušných XMLa souborů a byl spuštěn TestAut, který následně správně vyhodnocoval a nabízel řešení konfliktů. V prvním testu bylo odpovězeno *Ano* na všechny zobrazované konflikty. Druhý test probíhal za stejných podmínek, ale odpovězeno bylo *Ne* na všechny konflikty. Jak lze vidět v přílohách B.3 a B.4, přesun lokálního uzlu TEXTS_DEFINES byl proveden přesně podle výše zmíněných pravidel.

Dále bylo otestováno, že menu TestAutu již nenabízí možnost upravovat lokální Textové definice a že úprava serverových Textových definic probíhá v pořádku. Také bylo pomocí miniDOM_Monitoru ověřeno, že se po startu systému zkopírují Textové definice ze serveru do lokálního uzlu TEMPORARY/TEXTS_DEFINES. Posledním testem bylo ověřeno, že TestAut načítá Textové definice nejprve z tohoto dočasného uzlu a až v případě že zde Textovou definici nenalezne, pokusí se ji vyhledat na serveru. Test probíhal jednoduše tak, že po načtení TestAutu byla vybraná Textová definice na serveru změněna, ale TestAut poté stále zobrazoval při jejím použití její původní hodnotu, tedy dotazoval se do uzlu TEMPORARY/TEXTS_DEFINE. Toto chování se může zdát jako nedokonalé a může vést k nekonzistenci Textových definic mezi lokálním a serverovým XML, což je sice pravda, ale případné následky nejsou nikterak vážné, jelikož se jedná pouze o textové řetězce, které se navíc mění jen zřídka kdy.

7.1.2 Testování přesunu Nápovědných textů na server

Stejně jako se podobalo přesouvání Nápovědných textů na server přesunu Textových definic, tak i testování správnosti těchto procesů probíhalo podobně. I zde je možné narazit na 16 různých kombinací vztahů mezi lokálním a serverovým uzlem HELP_DEFINE. Opět byly sestaveny testovací uzly a byl nad jejich XML soubory spuštěn TestAut a TestAutServer. V první části testu byly všechny konflikty řešeny předností lokálních uzlů, ve druhé části byly všechny lokální uzly ignorovány. Tento testovací případ byl nejprve proveden pro nápovědy typu TEXT a poté byl zopakován pro nápovědy typu FILE. Ve druhém případě bylo navíc kontrolováno, zda se spolu s uzly přenáší na server i soubory, na které odkazují. Výsledky všech zmíněných testovacích případů byly pozitivní - uzly i případné soubory byly přeneseny dle očekávání. Na konci každého spuštění TestAutu byl úspěšně vytvořen a uložen .zip soubor s obsahem složky HtmlHelp, která byla následně úspěšně smazána.

Následoval test grafického uživatelského rozhraní, kde byly v dialogu Správce nápovědných textů správně zobrazeny pouze Nápovědné texty ze serveru. Při každé jejich úpravě byla změna provedena i na serveru a také byl ihned aktualizován lokální uzel TEMPORARY/HELP_LIST_DEFINE. Nakonec bylo otestováno i správné zobrazování a otevírání Nápovědných textů přímo od testů a parametrů, které na ně odkazují. Potvrdilo se, že výše popisovaný nedostatek byl vyřešen a Nápovědné texty se správně zobrazují nezávisle na tom, zda na ně odkazuje konkrétní uzel anebo některý z jeho rodičovských uzlů.

7.1.3 Testování přesunu definic testů na server

Během testování správnosti přesunu definic testů z lokálního XML souboru na server byly postupně nasimulovány všechny kroky, které tento přesun řeší. Ať už se jedná o změnu identifikátoru v případě kolize (včetně správnosti rozšíření tohoto identifikátoru na všechna místa, která se na daný test odkazují), nebo ať se jedná například o úpravu obsahu uzlu

HARDWARE_LIST v lokálním i serverovém XML. Každý z prováděných kroků byl nasimulován zvlášť použitím vhodné kombinace lokálního a serverového XML. Následoval test s využitím reálných XML souborů z pracovních počítačů. I u těchto výsledků bylo následně textovým komparátorem vyhodnoceno, zda provedené změny odpovídají požadavkům. V poslední části tohoto testování bylo ověřováno, že i po přesunu na server je zachována veškerá funkcionalita, kterou TestAut pro testy poskytuje. Jednalo se tak o provádění testovacích případů, jako jsou například tyto: Otevření, úprava a uložení parametrů testu; otevření, úprava a uložení atributů testu; vytváření nových testů a sekvencí; mazání testů a sekvencí; přesouvání testů a sekvencí v rámci levého podokna; přesouvání testů a sekvencí z levého podokna do pravého; a tak dále. Případné nalezené nedostatky byly ihned řešeny opravou ve zdrojových kódech. I když kvůli velkému množství potencionálních operací, které lze s testy provést, nebyly otestovány kompletně všechny možné situace, tak naprostá většina především těch běžně používaných byla do testování zahrnuta.

7.1.4 Testování přesunu uložených sekvencí testů na server

Přesun uložených sekvencí byl také otestován tak, aby byly ověřeny všechny funkce, které tento proces zahrnuje. Tento přesun není tak komplikovaný, jako tomu bylo v případě samotných definic testů. Poté, co bylo otestováno, že přesun je skutečně za každých podmínek úspěšný, byla testována práce s uloženými sekvencemi umístěnými již na serveru. Načítání, ukládání i mazání sekvencí testů probíhalo stejně, jako tomu bylo dříve, ovšem s tím rozdílem, že všechna data byla správně upravována v serverovém XML.

7.1.5 Testování přesunu hardwarových definic na server

Posledním z testovaných přesunů dat na server bylo přemisťování hardwarových definic. Správné řešení konfliktů identifikátorů, uložení identifikátorů lokálních hardwarových definic i samotný přesun všech uzlů HARDWARE_DEF proběhl ve všech testovacích případech úspěšně. Náročnější bylo následné testování správnosti používání hardwarových definic a jejich přepínání ze serverového XML souboru. Nejprve byla ověřena správná funkce blokování otevřeného hardwaru pro ostatní instance nástroje TestAut. Bylo otestováno, zda se po zvolení hardwaru ve vysouvací nabídce projeví tato změna ve všech oblastech nástroje - v testech zobrazených v levém podokně, v testech připravených v pravém podokně, v nabízených uložených sekvencích i v jednotlivých prvcích závislých na daném hardwaru, jako jsou zařízení, jednotky a tak dále. Všechny změny prováděné v těchto položkách se správně projevily pouze v příslušné definici hardwaru v serverovém XML souboru. Také bylo kontrolováno, zda se chování nástroje liší podle toho, zda je vybraný hardware lokální či nikoliv.

7.1.6 Testování Online a Offline módu

Testování Online a Offline módu nebylo příliš obtížné. Byla testována schopnost správně přijmout a rozpoznat parametry, se kterými je spouštěcí soubor otevřen a zda po spuštění s parametrem –offline bude skutečně TestAut spuštěn v příslušném módu. Dále bylo testováno přepínání mezi módy, a to jak pomocí tlačítka v dialogu Konfigurace, tak pomocí funkční klávesy F9. Bylo také testováno, zda je v Offline módu skutečně zabráněno ve vyhledávání připojených zařízení při startu TestAutu a při přepnutí vybraného hardwaru. Dále bylo testováno, zda v Offline módu skutečně nelze spustit testovací sekvence a ani

nelze ručním příkazem spustit vyhledávání zařízení. Průběžně bylo sledováno, zda byl vždy správně indikován Offline stav příznakem v titulku hlavního okna.

7.2 Integrační testy

Poslední část testování probíhala přímo v prostředí, kde se TestAut aktuálně využívá a to v počítačové síti testovací laboratoře společnosti Škoda Auto. Testování probíhalo ve spolupráci s kolegou Ing. Vladimirem Biathem, který s TestAutem pracuje v pozici Test administrátora. Byl vybrán jeden z počítačů určených pro integrační testy řídicích jednotek a na něm byly provedeny porovnávací testy mezi původní a nově vzniklou verzí TestAutu a poté byla testována samotná nová verze. První část se zabývala porovnáváním časů potřebných pro vykonání určité operace nad XML soubory. Byly zvoleny často používané operace, které mají největší vliv na komfort užívání nástroje. Testování nové verze nástroje probíhalo až poté, co byly na server nahrány všechny položky přesouvané na server pro všechny aktuálně používané lokální XML soubory z počítačů jednotlivých breadboardů. Testování probíhalo za stejných výchozích podmínek pro obě verze TestAutu, které byly postupně spuštěny na stejném vybraném počítači nad stejným lokálním XML souborem a připojovaly se přes síť k dalšímu počítači, na kterém byl spuštěn TestAutServer. I serverový počítač byl po celou dobu měření stejný a taktéž vytížení sítě bylo stejné, protože bylo testováno v době, kdy všechny ostatní počítače v rámci této laboratoře byly vypnuty a provoz mimo tuto síť byl prakticky nulový. Zajištění těchto stejných podmínek bylo důležité, protože naměřené časy jsou velmi závislé na síťové komunikaci mezi počítačem kde je spuštěn TestAut a počítačem, kde je spuštěn TestAutServer. Protože nová verze nástroje TestAut přesunula podstatnou část dat z lokálního XML souboru na server, bylo očekáváno, že reakční doba při zpracování XML dat se prodlouží. Výsledky měření jsou uvedeny v tabulce 7.1.

Výsledky měření ukazují, že časy spouštění TestAutu a načítání položek hardwaru po přepnutí ve vysouvací nabídce se sice prodloužily, ale tento nárůst není nikterak markantní a v práci nebrání. Načítání uložených sekvencí testů bylo prováděno nad pěti nejčastěji používanými sekvencemi testů, které jsou na vybraném počítači spouštěny. Zde se již plně potvrdil předpoklad, že stahování většího množství položek ze serverového XML výrazně prodlouží dobu provedení některých operací. Ještě výrazněji se tento předpoklad projevil během načítání ukládacího dialogu pro uložení parametru testu, kde se zobrazuje strom všech uzlů testů, které mají s právě zpracovávaným testem společného předka. Tento kompletní přehled o dědičnosti testů napříč všemi breadboardy, je jednou z očekávaných výhod, jenž nové umístění dat na serveru přinese. Nárůst času potřebného pro zobrazení ukládacího dialogu je přímo úměrný počtu testů, které jsou s právě zpracovávaným testem příbuzné. Proto se tak výrazně liší výsledky pro jednotlivé parametry zvolené pro testování. Jak je vidět na obrázku 3.4, lze ve stromu zobrazovaném při ukládání parametru testu vynechat testy z uložených sekvencí. Tím se sníží počet zobrazovaných uzlů stromu a také potřebná doba načítání v závislosti na tom, kolik z příbuzných testů je uloženo právě v uložených sekvencích. Testování probíhalo se zobrazováním i těchto testů. Po jejich vynechání se nárůst času zobrazený v tabulce 7.1 snížil přibližně na polovinu.

Kromě právě popsaného měření byla hodnocena i rychlost odezvy během jednodušších úkonů, jako například zobrazení parametrů testu, otevření parametru testu, přidání parametru testu či úprava atributů testu. Časová odezva tohoto typu operací nebyla nijak výrazně prodloužena a zůstala i nadále tak nízká, že se dané operace provádí v reálném čase.

Testovaná operace	Čas původní verze [s]	Čas nové verze [s]	Nárůst času u nové verze oproti verzi původní [%]
Spouštění TestAutu (bez vyhledávání zařízení)	21	27	28,6
Načtení hardwaru po přepnutí ve vysouvací nabídce	14	20	42,9
Načítání sekvence testů č. 1	22	80	263,6
Načítání sekvence testů č. 2	23	89	287,0
Načítání sekvence testů č. 3	15	55	266,7
Načítání sekvence testů č. 4	11	47	327,3
Načítání sekvence testů č. 5	3	14	366,7
Načtení ukládacího dialogu pro parametr č. 1	16	56	250
Načtení ukládacího dialogu pro parametr č. 2	8	40	400
Načtení ukládacího dialogu pro parametr č. 3	2	20	900
Načtení ukládacího dialogu pro parametr č. 4	10	114	1040

Tabulka 7.1: Porovnání průměrných časů potřebných k provedení často používaných operací s nástrojem TestAut mezi původní a novou verzí nástroje.

Další částí celkového testování nové verze nástroje TestAut bylo provádění testovacích případů, které byly vybrány tak, aby otestovaly co nejvíce z používaných funkcí TestAutu. Jedná se například o tyto případy:

- Spuštění TestAutu, přepnutí do administrátorského módu, přepnutí na jiný lokální hardware, načtení uložené sekvence, spuštění sekvence testů.
- Spuštění TestAutu, přepnutí do administrátorského módu, přepnutí na hardware variantu UNIVERSAL, pokus o načtení uložených sekvencí testů.
- Spuštění TestAutu, vytvoření sekvence testů přetažením testů z levého podokna, lokální úprava parametrů testů na pravé straně, uložení sekvence testů.
- Spuštění TestAutu, přepnutí do administrátorského módu, vytvoření nového testu pro aktuální hardware, následná změna hardwaru testu na jiný.
- Spuštění TestAutu, úprava parametru testu typu reference na parametr, která odkazuje do serverového uzlu ROOT/PARAMETERS, následně přesun tohoto testu do pravého podokna.
- Spuštění TestAutu v Offline módu, načtení uložené sekvence testů, pokus o spuštění testovací sekvence.
- Spuštění TestAutu v Offline módu, načtení uložené sekvence testů, přepnutí do Online módu, spuštění testovací sekvence.

Pro každý testovací případ bylo ověřeno, zda TestAut provedl všechny požadované kroky a bylo dosaženo očekávaného stavu. V několika případech nebylo očekávaného stavu dosaženo a tyto nedokonalosti byly následně opraveny úpravou zdrojových kódů.

7.3 Zhodnocení dosažených výsledků

Na základě informací uvedených výše v této kapitole, lze říci, že cílů stanovených v kapitole 4 bylo dosaženo. Navržené principy a použitá řešení umožnila provedení požadovaných úprav, někdy ale přinesla i nová omezení. Řeč je především o prodloužení reakčních časů složitějších operací způsobených nutností pracovat více se serverovým XML souborem. Dle vyjádření Ing. Vladimira Biatha, který je nejčastějším uživatelem nástroje TestAut a pro jehož práci je důležité mít možnost spravovat co největší část informací z XML souborů na jednom místě, však přínosy nového systému převyšují nad vzniklými nevýhodami. Následuje citace jeho vyjádření po zhlédnutí prezentace nové verze TestAutu od autora této práce a po společném testování nové verze TestAutu:

> Na základě přidané funkcionality přepínání se mezi jednotlivými projekty v testovacím programu TestAut, odpadá potřeba ukončení, respektive zavření, již spuštěného TestAutu s daným projektem. Tato nová úprava mně jako funkčnímu uživateli výrazným způsobem usnadňuje práci z hlediska okamžité změny projektu a úspory času při úpravě jednotlivých testů. Další výhodu této úpravy vidím v jednoduchosti změny projektu. Získaný čas navíc je možné využít na jiné věci, ať už to souvisí přímo či nepřímo s TestAutem.

Volbu toho, která vylepšení implementovat do nástroje TestAut, lze považovat za správnou, protože všechna přidaná vylepšení se po otestování jeví jako užitečná a budou využívána při běžném provozu nástroje.

V době odevzdání této práce je situace taková, že nová verze nástroje TestAut je nasazena na jednom z počítačů určených k integračnímu testování řídicích jednotek vozidel a je zde používána v testovacím (pilotním) provozu. Protože ani výše popsaným testováním nebyly jistě odstraněny kompletně všechny chyby, bude TestAut nasazen na zbývající počítače až po určitém čase - až se praktickým používáním tohoto nástroje dostatečně osvědčí jeho dlouhodobá stabilita a odladí všechny případné další nalezené chyby.

Kapitola 8

Závěr

Cílem této práce bylo analyzovat testovací nástroje pro automatizované integrační testování elektronických řídicích jednotek automobilů. Hlavní důraz byl kladen na nástroj TestAut 2, který je vyvíjen společností e4t a slouží pro testování automobilů značky Škoda. Tento nástroj je podrobně popsán v podkapitole 3.1 a to jak z hlediska jeho vnitřní stavby, tak z hlediska použití uživateli. TestAut 2 sice splňuje základní nároky, které jsou na něj kladené, ale zároveň existuje velký prostor pro zlepšení a to jak z hlediska funkčnosti, tak z pohledu usnadnění jeho obsluhy uživatelům. Tato práce měla prozkoumat konkurenční nástroje používané ke stejným účelům a vybrat některé klíčové vlastnosti, které se zdají být pro oblast integračního testování ŘJ podstatné. Analyzován byl nástroj EXAM, který je používán ve společnosti e4t k jiným testovacím účelům. Stejně tak nástroj MODENA, který je ve společnosti e4t používán například pro testování ŘJ autorádií. Posledním analyzovaným nástrojem byl systém PROVEtech, který vyvíjí konkurenční společnost MBtech.

Jako klíčové principy pro testovací nástroje v této oblasti průmyslu byly zvoleny: architektura klient-server, Online a Offline mód, vzdálené spuštění testu, použití databáze pro uložení testů a tvorba testů formou UML diagramů. Popis těchto principů, odůvodnění jejich volby a porovnání se stavem nástroje TestAut 2 je v podkapitole 4. Tato práce dále popisuje v kapitole 5 návrh úprav stávajícího testovacího systému tak, aby splňoval vybrané principy vhodné pro nasazení na integrační testy řídicích jednotek vozů ve Škoda Auto a.s. Jedná se konkrétně o lepší využití architektury klient-server a o vytvoření Online a Offline módu aplikace. Kapitola 6 pojednává o samotném provedení těchto úprav ve stávajícím zdrojovém kódu. Veškeré provedené změny byly řádně otestovány a výsledky testů byly diskutovány v kapitole 7.

Výsledkem této práce je nová verze testovacího nástroje TestAut 2, vylepšená o výše zmíněné funkce, která byla otestována a následně také nasazena k použití v praxi na skutečném breadboardu. Nové funkce byly uživateli nástroje vyhodnoceny jako přínosné, ačkoliv s sebou nesou i omezení ve formě pomalejší odezvy některých operací. Autor této práce navrhuje jako budoucí pokračování práce řešit toto omezení úpravou použitého XML parseru (MiniDOM serveru). V současném stavu totiž poskytuje pouze jednoduché funkce a veškeré složitější algoritmy nad XML dokumenty tak probíhají až v rámci nástroje TestAut 2 za použití velkého množství těchto jednoduchých příkazů, což přináší značnou režii, především při práci s XML dokumentem přes počítačovou síť. Autorem navrhované vylepšení tkví v přenesení složitosti funkcí pro práci s XML dokumentem na tento parser, který by tak nástroji TestAut 2 poskytoval komplexnější odpovědi již po jediném dotazu, což by výrazně snížilo režii při práci se vzdáleným XML dokumentem a snížilo tak výrazně dobu odezvy operací prováděných v nástroji TestAut 2.

Literatura

[1] *About MicroNova*. [Online; navštíveno 8.1.2016].
URL http://www.micronova.de/

[2] *A comprehensive tool for test automation*. [Online; navštíveno 3.1.2016].
URL https://www.mbtech-group.com/cz/electronics_solutions/tools_equipment/provetechta_test_automation.html

[3] *Bus Systems*. [Online; navštíveno 8.1.2016].
URL http://automotive.softing.com/en/standards/bus-systems.html

[4] *eclipse*. [Online; navštíveno 8.1.2016].
URL http://www.eclipse.org/

[5] *EXAM*. [Online; navštíveno 8.1.2016].
URL
https://www.exam-ta.de/images/stories/exam/PI_EXAM_EN_1v0_digital.pdf

[6] *EXAM Concept paper*. [Online; navštíveno 8.1.2016].
URL https://www.exam-ta.de/en/downloads/category/4-dokumente/98-exam_conceptpaper.html

[7] *EXAM is ...* [Online; navštíveno 8.1.2016].
URL https://www.exam-ta.de

[8] *MBtech group*. [Online; navštíveno 3.1.2016].
URL https://www.mbtech-group.com/

[9] *MFC Desktop Applications*. [Online; navštíveno 21.12.2015].
URL https://msdn.microsoft.com/en-us/library/d06h2x6e.aspx

[10] *Microsoft .NET*. [Online; navštíveno 3.1.2016].
URL https://www.microsoft.com/net

[11] *MODENA*. [Online; navštíveno 8.1.2016].
URL http://www.berner-mattner.com/en/products/modena/index.html

[12] *MODENA Architecture*. [Online; navštíveno 8.1.2016].
URL http://automotive-eetimes.com/images/01-edit-photo-uploads/2010/2010-11-november/modena_grafik_300dpi.jpg

[13] *PROVETECH NEWSLETTER*. [Online; navštíveno 3.1.2016].
URL https://www.mbtech-group.com/fileadmin/media/pdf/electronics_solutions/141216_PROVEtech-Newsletter_2-2014_EN.pdf

[14] *PROVEtech NEWSLETTER*. [Online; navštíveno 3.1.2016].
URL https://www.mbtech-group.com/fileadmin/media/pdf/electronics_solutions/PROVEtech-Newsletter_01-2015_EN.pdf

[15] *PROVEtech NEWSLETTER 1/2014*. [Online; navštíveno 3.1.2016].
URL https://www.mbtech-group.com/fileadmin/media/pdf/electronics_solutions/140528_PROVEtech-Newsletter_1-2014_EN.pdf

[16] *PROVEtech NEWSLETTER 2/2013*. [Online; navštíveno 3.1.2016].
URL https://www.mbtech-group.com/fileadmin/media/pdf/electronics_solutions/PROVEtech-Newsletter_2-2013_EN.pdf

[17] *PROVEtech:TA - Diagnostics*. [Online; navštíveno 3.1.2016].
URL https://www.mbtech-group.com/cz/electronics_solutions/tools_equipment/provetechta_test_automation/diagnostics.html

[18] *PROVEtech:TA - Fault Simulation*. [Online; navštíveno 3.1.2016].
URL https://www.mbtech-group.com/cz/electronics_solutions/tools_equipment/provetechta_test_automation/fault_simulation.html

[19] *PROVEtech:TA RTT+ - Model-based testing*. [Online; navštíveno 3.1.2016].
URL https://www.mbtech-group.com/eu-en/electronics_solutions/tools_equipment/provetechta_test_automation/model_based_testing.html

[20] *PROVEtech:TA - Testmanager*. [Online; navštíveno 3.1.2016].
URL https://www.mbtech-group.com/cz/electronics_solutions/tools_equipment/provetechta_test_automation/testmanager.html

[21] *PROVEtech:TA: The comprehensive tool for test automation*. [Online; navštíveno 3.1.2016].
URL https://www.mbtech-group.com/fileadmin/media/pdf/electronics_solutions/140124_leaflet_PROVEtech-TA_EN_dsc_Web.pdf

[22] *PROVEtech:TA - Workspace*. [Online; navštíveno 3.1.2016].
URL https://www.mbtech-group.com/cz/electronics_solutions/tools_equipment/provetechta_test_automation/workspace.html

[23] *PROVEtech tool suite*. [Online; navštíveno 3.1.2016].
URL https://www.mbtech-group.com/cz/electronics_solutions/tools_equipment/provetech_tool_suite.html

[24] *python*. [Online; navštíveno 8.1.2016].
URL https://www.python.org/

[25] *Rational Rhapsody Architect for Software*. [Online; navštíveno 8.1.2016].
URL http://www-03.ibm.com/software/products/en/ratirhaparchforsoft

[26] *Scintilla*. [Online; navštíveno 21.12.2015].
URL http://www.scintilla.org/

[27] *Scripting with WinWrap Basic*. [Online; navštíveno 3.1.2016].
URL https://www.winwrap.com/

[28] *Tcl Developer Site*. [Online; navštíveno 21.12.2015].
URL https://www.tcl.tk/

[29] *Tcl/Tk Audience*. [Online; navštíveno 21.12.2015].
URL https://www.tcl.tk/about/audience.html

[30] *Verified Systems International GmbH*. [Online; navštíveno 3.1.2016].
URL https://www.verified.de/

[31] *Vymýšlíme auta, která budou online, říká vývojář Škoda Auto*. [Online; navštíveno 8.1.2016].
URL
http://www.jobs.cz/poradna/vyvojar-vymyslime-auta-ktera-budou-online/

[32] *Welcome To CORBA Web Site!* [Online; navštíveno 8.1.2016].
URL http://www.corba.org/

[33] *Welcome to the AUTOSAR development partnership*. [Online; navštíveno 3.1.2016].
URL http://www.autosar.org/

[34] *What is SymmetricDS*. [Online; navštíveno 3.1.2016].
URL http://www.symmetricds.org/

[35] *CAN Specification*. 1991, [Online; navštíveno 1.9.2015].
URL http://www.bosch-semiconductors.de/media/ubk_semiconductors/pdf_1/canliteratur/can2spec.pdf

[36] [ed.], W. L.: *CAN System Engineering : From Theory to Practical Applications*. Springer London, 2013, ISBN 978-1-4471-5613-0.

[37] Huang, Y.; McMurran, R.; Amor-Segan, M.; aj.: *Development of an automated testing system for vehicle infotainment system. The International Journal of Advanced Manufacturing Technology*, ročník 51, č. 1, 2010: s. 233–246, ISSN 1433-3015.

[38] Kaiser, M.; Reuschenbach, L.; Scheible, B.; aj.: *ECU development. Bosch Professional Automotive Information*, 2014: s. 326–343.

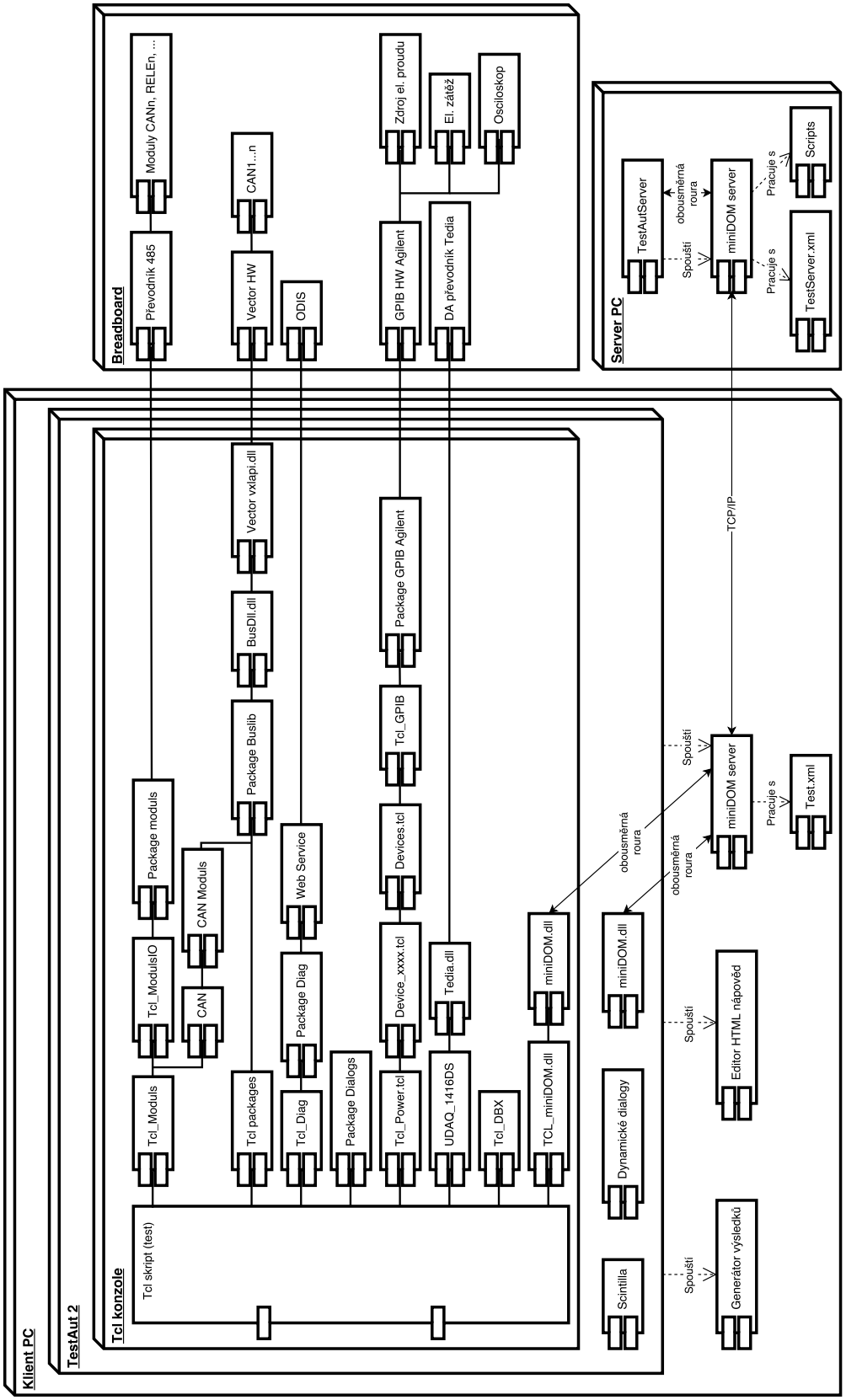
[39] Kiencke, U.; Dais, S.; Litschel, M.: *Automotive Serial Controller Area Network. SAE Technical Paper*, 1986, ISSN 0148-7191.

Přílohy

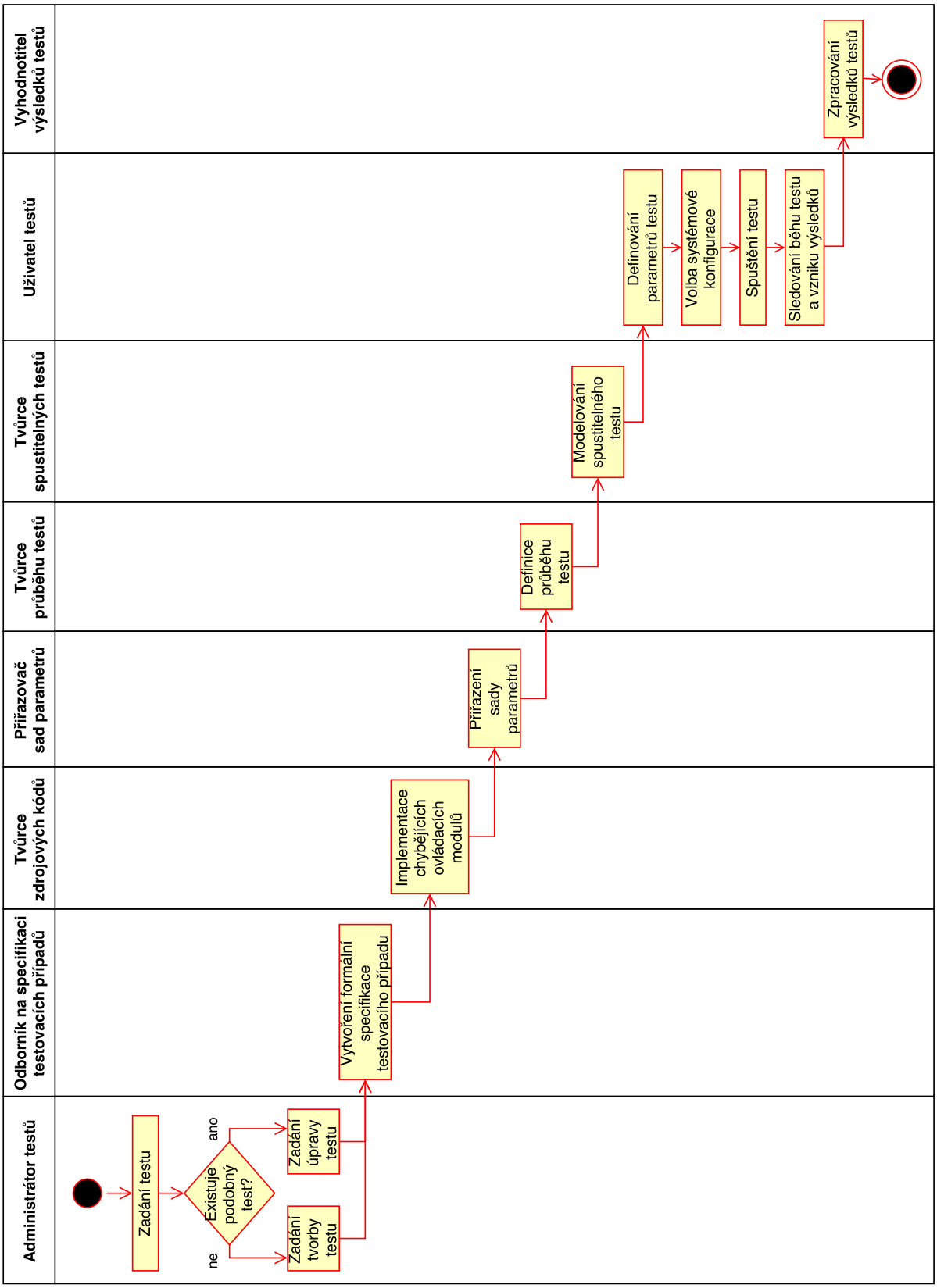
Seznam příloh

Příloha A

Diagramy



Obrázek A.1: Diagram nasazení nástroje TestAut.



Obrázek A.2: Diagram aktivit při použití nástroje EXAM.

Příloha B

Zdrojové kódy

```
<TEXTS_DEFINE>
   <TEXT_DEFINE ID="ID_SHODA">
      <TEXT LANG="CZ">shoda</TEXT>
   </TEXT_DEFINE>
   <TEXT_DEFINE ID="ID_KONFKLIKT">
      <TEXT LANG="CZ">konflikt</TEXT>
   </TEXT_DEFINE>
   <TEXT_DEFINE ID="ID_JEN_LOKALNE">
      <TEXT LANG="CZ">jen lokalne</TEXT>
   </TEXT_DEFINE>
   <TEXT_DEFINE ID="ID_PRAZDNA_LOKALNE"/>
   <TEXT_DEFINE ID="ID_KONFKLIKT_A_SHODA">
      <TEXT LANG="CZ">konflikt</TEXT>
      <TEXT LANG="EN">shoda</TEXT>
   </TEXT_DEFINE>
   <TEXT_DEFINE ID="ID_JEN_LOKAL_A_SHODA">
      <TEXT LANG="CZ">jen lokalne</TEXT>
      <TEXT LANG="EN">shoda</TEXT>
   </TEXT_DEFINE>
   <TEXT_DEFINE ID="ID_JEN_LOKAL_A_KONFLIKT">
      <TEXT LANG="CZ">jen lokalne</TEXT>
      <TEXT LANG="EN">konflikt</TEXT>
   </TEXT_DEFINE>
   <TEXT_DEFINE ID="ID_JEN_SERVER_A_KONFLIKT">
      <TEXT LANG="EN">konflikt</TEXT>
   </TEXT_DEFINE>
   <TEXT_DEFINE ID="ID_JEN_SERVER_A_SHODA">
      <TEXT LANG="EN">shoda</TEXT>
   </TEXT_DEFINE>
   <TEXT_DEFINE ID="ID_JEN_SERVER_A_JEN_LOKAL">
      <TEXT LANG="CZ">jen lokalne</TEXT>
   </TEXT_DEFINE>
   <TEXT_DEFINE ID="ID_KONFKLIKT_A_SHODA_A_JEN_LOKAL">
      <TEXT LANG="CZ">konflikt</TEXT>
      <TEXT LANG="EN">shoda</TEXT>
      <TEXT LANG="DE">jen lokalne</TEXT>
   </TEXT_DEFINE>
   <TEXT_DEFINE ID="ID_KONFKLIKT_A_SHODA_A_JEN_SERVER">
      <TEXT LANG="CZ">konflikt</TEXT>
```

```
40        <TEXT LANG="EN">shoda</TEXT>
41     </TEXT_DEFINE>
42     <TEXT_DEFINE ID="ID_JEN_SERVER_A_JEN_LOKAL_A_KONFLIKT">
43        <TEXT LANG="CZ">jen lokalne</TEXT>
44        <TEXT LANG="EN">konflikt</TEXT>
45     </TEXT_DEFINE>
46     <TEXT_DEFINE ID="ID_JEN_SERVER_A_JEN_LOKAL_A_SHODA">
47        <TEXT LANG="CZ">jen lokalne</TEXT>
48        <TEXT LANG="EN">shoda</TEXT>
49     </TEXT_DEFINE>
50     <TEXT_DEFINE ID="ID_JEN_SERVER_JEN_LOKAL_A_SHODA_A_KONFLIKT">
51        <TEXT LANG="CZ">jen lokalne</TEXT>
52        <TEXT LANG="EN">shoda</TEXT>
53        <TEXT LANG="DE">konflikt</TEXT>
54     </TEXT_DEFINE>
55 </TEXTS_DEFINE>
```

Zdrojový kód B.1: Lokální uzel TEXTS_DEFINE pro test přesunu na server

```
1  <TEXTS_DEFINE VERSION="100">
2     <TEXT_DEFINE ID="ID_SHODA">
3        <TEXT LANG="CZ">shoda</TEXT>
4     </TEXT_DEFINE>
5     <TEXT_DEFINE ID="ID_KONFKLIKT">
6        <TEXT LANG="CZ">k o n f l i k t</TEXT>
7     </TEXT_DEFINE>
8     <TEXT_DEFINE ID="ID_JEN_SERVER">
9        <TEXT LANG="CZ">jen server</TEXT>
10    </TEXT_DEFINE>
11    <TEXT_DEFINE ID="ID_PRAZDNA_SERVER"/>
12    <TEXT_DEFINE ID="ID_KONFKLIKT_A_SHODA">
13       <TEXT LANG="CZ">k o n f l i k t</TEXT>
14       <TEXT LANG="EN">shoda</TEXT>
15    </TEXT_DEFINE>
16    <TEXT_DEFINE ID="ID_JEN_LOKAL_A_SHODA">
17       <TEXT LANG="EN">shoda</TEXT>
18    </TEXT_DEFINE>
19    <TEXT_DEFINE ID="ID_JEN_LOKAL_A_KONFLIKT">
20       <TEXT LANG="EN">k o n f l i k t</TEXT>
21    </TEXT_DEFINE>
22    <TEXT_DEFINE ID="ID_JEN_SERVER_A_KONFLIKT">
23       <TEXT LANG="CZ">jen server</TEXT>
24       <TEXT LANG="EN">k o n f l i k t</TEXT>
25    </TEXT_DEFINE>
26    <TEXT_DEFINE ID="ID_JEN_SERVER_A_SHODA">
27       <TEXT LANG="CZ">jen server</TEXT>
28       <TEXT LANG="EN">shoda</TEXT>
29    </TEXT_DEFINE>
30    <TEXT_DEFINE ID="ID_JEN_SERVER_A_JEN_LOKAL">
31       <TEXT LANG="EN">jen server</TEXT>
32    </TEXT_DEFINE>
33    <TEXT_DEFINE ID="ID_KONFKLIKT_A_SHODA_A_JEN_LOKAL">
34       <TEXT LANG="CZ">k o n f l i k t</TEXT>
35       <TEXT LANG="EN">shoda</TEXT>
```

```
36     </TEXT_DEFINE>
37     <TEXT_DEFINE ID="ID_KONFKLIKT_A_SHODA_A_JEN_SERVER">
38        <TEXT LANG="CZ">k o n f l i k t</TEXT>
39        <TEXT LANG="EN">shoda</TEXT>
40        <TEXT LANG="DE">jen server</TEXT>
41     </TEXT_DEFINE>
42     <TEXT_DEFINE ID="ID_JEN_SERVER_A_JEN_LOKAL_A_KONFLIKT">
43        <TEXT LANG="DE">jen server</TEXT>
44        <TEXT LANG="EN">k o n f l i k t</TEXT>
45     </TEXT_DEFINE>
46     <TEXT_DEFINE ID="ID_JEN_SERVER_A_JEN_LOKAL_A_SHODA">
47        <TEXT LANG="DE">jen server</TEXT>
48        <TEXT LANG="EN">shoda</TEXT>
49     </TEXT_DEFINE>
50     <TEXT_DEFINE ID="ID_JEN_SERVER_JEN_LOKAL_A_SHODA_A_KONFLIKT">
51        <TEXT LANG="SK">jen server</TEXT>
52        <TEXT LANG="EN">shoda</TEXT>
53        <TEXT LANG="DE">k o n f l i k t</TEXT>
54     </TEXT_DEFINE>
55 </TEXTS_DEFINE>
```

Zdrojový kód B.2: Vzdálený uzel TEXTS_DEFINE pro test přesunu na server

```
1  <TEXTS_DEFINE VERSION="101">
2     <TEXT_DEFINE ID="ID_SHODA">
3        <TEXT LANG="CZ">shoda</TEXT>
4     </TEXT_DEFINE>
5     <TEXT_DEFINE ID="ID_KONFKLIKT">
6        <TEXT LANG="CZ">konflikt</TEXT>
7     </TEXT_DEFINE>
8     <TEXT_DEFINE ID="ID_JEN_SERVER">
9        <TEXT LANG="CZ">jen server</TEXT>
10    </TEXT_DEFINE>
11    <TEXT_DEFINE ID="ID_PRAZDNA_SERVER"/>
12    <TEXT_DEFINE ID="ID_KONFKLIKT_A_SHODA">
13       <TEXT LANG="CZ">konflikt</TEXT>
14       <TEXT LANG="EN">shoda</TEXT>
15    </TEXT_DEFINE>
16    <TEXT_DEFINE ID="ID_JEN_LOKAL_A_SHODA">
17       <TEXT LANG="CZ">jen lokalne</TEXT>
18       <TEXT LANG="EN">shoda</TEXT>
19    </TEXT_DEFINE>
20    <TEXT_DEFINE ID="ID_JEN_LOKAL_A_KONFLIKT">
21       <TEXT LANG="CZ">jen lokalne</TEXT>
22       <TEXT LANG="EN">konflikt</TEXT>
23    </TEXT_DEFINE>
24    <TEXT_DEFINE ID="ID_JEN_SERVER_A_KONFLIKT">
25       <TEXT LANG="CZ">jen server</TEXT>
26       <TEXT LANG="EN">konflikt</TEXT>
27    </TEXT_DEFINE>
28    <TEXT_DEFINE ID="ID_JEN_SERVER_A_SHODA">
29       <TEXT LANG="CZ">jen server</TEXT>
30       <TEXT LANG="EN">shoda</TEXT>
31    </TEXT_DEFINE>
```

```
32      <TEXT_DEFINE ID="ID_JEN_SERVER_A_JEN_LOKAL">
33         <TEXT LANG="EN">jen server</TEXT>
34         <TEXT LANG="CZ">jen lokalne</TEXT>
35      </TEXT_DEFINE>
36      <TEXT_DEFINE ID="ID_KONFKLIKT_A_SHODA_A_JEN_LOKAL">
37         <TEXT LANG="CZ">konflikt</TEXT>
38         <TEXT LANG="EN">shoda</TEXT>
39         <TEXT LANG="DE">jen lokalne</TEXT>
40      </TEXT_DEFINE>
41      <TEXT_DEFINE ID="ID_KONFKLIKT_A_SHODA_A_JEN_SERVER">
42         <TEXT LANG="DE">jen server</TEXT>
43         <TEXT LANG="CZ">konflikt</TEXT>
44         <TEXT LANG="EN">shoda</TEXT>
45      </TEXT_DEFINE>
46      <TEXT_DEFINE ID="ID_JEN_SERVER_A_JEN_LOKAL_A_KONFLIKT">
47         <TEXT LANG="DE">server</TEXT>
48         <TEXT LANG="CZ">jen lokalne</TEXT>
49         <TEXT LANG="EN">konflikt</TEXT>
50      </TEXT_DEFINE>
51      <TEXT_DEFINE ID="ID_JEN_SERVER_A_JEN_LOKAL_A_SHODA">
52         <TEXT LANG="DE">jen server</TEXT>
53         <TEXT LANG="CZ">jen lokalne</TEXT>
54         <TEXT LANG="EN">shoda</TEXT>
55      </TEXT_DEFINE>
56      <TEXT_DEFINE ID="ID_JEN_SERVER_JEN_LOKAL_A_SHODA_A_KONFLIKT">
57         <TEXT LANG="SK">jen server</TEXT>
58         <TEXT LANG="CZ">jen lokalne</TEXT>
59         <TEXT LANG="EN">shoda</TEXT>
60         <TEXT LANG="DE">konflikt</TEXT>
61      </TEXT_DEFINE>
62      <TEXT_DEFINE ID="ID_JEN_LOKALNE">
63         <TEXT LANG="CZ">jen lokalne</TEXT>
64      </TEXT_DEFINE>
65      <TEXT_DEFINE ID="ID_PRAZDA_LOKALNE"/>
66  </TEXTS_DEFINE>
```

Zdrojový kód B.3: Vzdálený uzel TEXTS_DEFINE po testu přesunu na server při odpovědi Ano na všechny konflikty

```
1  <TEXTS_DEFINE VERSION="101">
2      <TEXT_DEFINE ID="ID_SHODA">
3         <TEXT LANG="CZ">shoda</TEXT>
4      </TEXT_DEFINE>
5      <TEXT_DEFINE ID="ID_KONFKLIKT">
6         <TEXT LANG="CZ">k o n f l i k t</TEXT>
7      </TEXT_DEFINE>
8      <TEXT_DEFINE ID="ID_JEN_SERVER">
9         <TEXT LANG="CZ">jen server</TEXT>
10     </TEXT_DEFINE>
11     <TEXT_DEFINE ID="ID_PRAZDNA_SERVER"/>
12     <TEXT_DEFINE ID="ID_KONFKLIKT_A_SHODA">
13        <TEXT LANG="CZ">k o n f l i k t</TEXT>
14        <TEXT LANG="EN">shoda</TEXT>
15     </TEXT_DEFINE>
```

```
16      <TEXT_DEFINE ID="ID_JEN_LOKAL_A_SHODA">
17         <TEXT LANG="CZ">jen lokalne</TEXT>
18         <TEXT LANG="EN">shoda</TEXT>
19      </TEXT_DEFINE>
20      <TEXT_DEFINE ID="ID_JEN_LOKAL_A_KONFLIKT">
21         <TEXT LANG="EN">k o n f l i k t</TEXT>
22      </TEXT_DEFINE>
23      <TEXT_DEFINE ID="ID_JEN_SERVER_A_KONFLIKT">
24         <TEXT LANG="CZ">jen server</TEXT>
25         <TEXT LANG="EN">k o n f l i k t</TEXT>
26      </TEXT_DEFINE>
27      <TEXT_DEFINE ID="ID_JEN_SERVER_A_SHODA">
28         <TEXT LANG="CZ">jen server</TEXT>
29         <TEXT LANG="EN">shoda</TEXT>
30      </TEXT_DEFINE>
31      <TEXT_DEFINE ID="ID_JEN_SERVER_A_JEN_LOKAL">
32         <TEXT LANG="EN">jen server</TEXT>
33         <TEXT LANG="CZ">jen lokalne</TEXT>
34      </TEXT_DEFINE>
35      <TEXT_DEFINE ID="ID_KONFKLIKT_A_SHODA_A_JEN_LOKAL">
36         <TEXT LANG="CZ">k o n f l i k t</TEXT>
37         <TEXT LANG="EN">shoda</TEXT>
38      </TEXT_DEFINE>
39      <TEXT_DEFINE ID="ID_KONFKLIKT_A_SHODA_A_JEN_SERVER">
40         <TEXT LANG="CZ">k o n f l i k t</TEXT>
41         <TEXT LANG="EN">shoda</TEXT>
42         <TEXT LANG="DE">jen server</TEXT>
43      </TEXT_DEFINE>
44      <TEXT_DEFINE ID="ID_JEN_SERVER_A_JEN_LOKAL_A_KONFLIKT">
45         <TEXT LANG="DE">jen server</TEXT>
46         <TEXT LANG="EN">k o n f l i k t</TEXT>
47      </TEXT_DEFINE>
48      <TEXT_DEFINE ID="ID_JEN_SERVER_A_JEN_LOKAL_A_SHODA">
49         <TEXT LANG="DE">jen server</TEXT>
50         <TEXT LANG="CZ">jen lokalne</TEXT>
51         <TEXT LANG="EN">shoda</TEXT>
52      </TEXT_DEFINE>
53      <TEXT_DEFINE ID="ID_JEN_SERVER_JEN_LOKAL_A_SHODA_A_KONFLIKT">
54         <TEXT LANG="SK">jen server</TEXT>
55         <TEXT LANG="EN">shoda</TEXT>
56         <TEXT LANG="DE">k o n f l i k t</TEXT>
57      </TEXT_DEFINE>
58      <TEXT_DEFINE ID="ID_JEN_LOKALNE">
59         <TEXT LANG="CZ">jen lokalne</TEXT>
60      </TEXT_DEFINE>
61      <TEXT_DEFINE ID="ID_PRAZDA_LOKALNE"/>
62  </TEXTS_DEFINE>
```

Zdrojový kód B.4: Vzdálený uzel TEXTS_DEFINE po testu přesunu na server při odpovědi Ne na všechny konflikty

Příloha C

Obsah DVD

- adresář **TestAut_old** se zdrojovými kódy nástroje TestAut ve verzi, která byla vstupem pro tuto diplomovou práci
- adresář **TestAut_new** se zdrojovými kódy nástroje TestAut ve verzi, která je výstupem této diplomové práce (obsahuje změny popisované v textu práce)
- adresář **TestAut_demo_local** s ukázkovými soubory potřebnými pro demonstrační spuštění lokální instance nové verze nástroje TestAut
- adresář **TestAut_demo_server** s ukázkovými soubory potřebnými pro demonstrační spuštění serverové instance nástroje TestAutServer
- adresář **tex** s textem této práce ve formátu .pdf i ve formě zdrojových kódů formátu .tex a dalších potřebných pro vytvoření pdf souboru
- soubor **license.txt** definující jaká licence se vztahuje na všechny zdrojové kódy odevzdávané v rámci této diplomové práce