VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV RADIOELEKTRONIKY

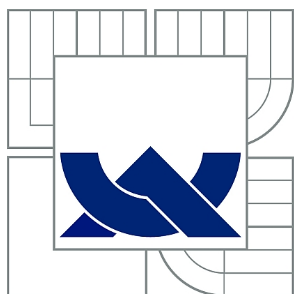
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF RADIO ELECTRONICS

AUTONOMNÍ GENERÁTOR TESTOVACÍCH SKRIPTŮ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

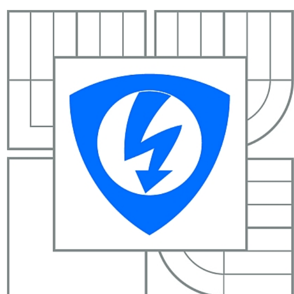
AUTOR PRÁCE Bc. STANISLAV HORKÝ
AUTHOR

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV RADIOELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF RADIO ELECTRONICS

AUTONOMNÍ GENERÁTOR TESTOVACÍCH SKRIPTŮ

AUTONOMOUS GENERATOR OF TEST SCRIPTS

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

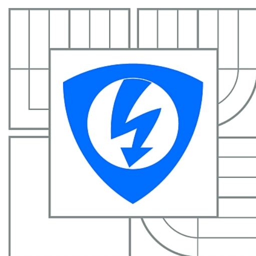
AUTOR PRÁCE Bc. STANISLAV HORKÝ

AUTHOR

VEDOUCÍ PRÁCE doc. Ing. JAROMÍR KOLOUCH, CSc.

SUPERVISOR

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav radioelektroniky

Diplomová práce

magisterský navazující studijní obor
Elektronika a sdělovací technika

Student: Bc. Stanislav Horký ***ID:*** 136521
Ročník: 2 ***Akademický rok:*** 2014/2015

NÁZEV TÉMATU:

Autonomní generátor testovacích skriptů

POKYNY PRO VYPRACOVÁNÍ:

Navrhněte koncept aplikace pro automatizované testování, která na základě předloženého stavového diagramu výrobku samostatně provede jeho funkční testování s pomocí I/O modulů. Navrhněte formát vstupních dat pro aplikaci a vytvořte generátor/editor vstupních dat. Aplikace bude vytvořena v prostředí C#.NET.
Vytvořte aplikaci podle navrženého konceptu. Zaměřte se na možnost použití aplikace s různými I/O moduly. Funkčnost aplikace demonstrujte aspoň na jednom projektu ve vývojovém centru firmy Honeywell.

DOPORUČENÁ LITERATURA:

[1] ČÁPKA, D. Úvod do WPF (Windows Presentation foundation) [online]. [cit. 2014-05-09]. Dostupné na www: http://www.devbook.cz/c-sharp-tutorial-wpf-uvod-a-prvni-formularova-aplikace.
[2] Visual C#.NET Programming [online]. [cit. 2014-05-09]. Dostupné na www: http://www.homeandlearn.co.uk/csharp/csharp.html.

Termín zadání: 9.2.2015 ***Termín odevzdání:*** 21.5.2015

Vedoucí práce: doc. Ing. Jaromír Kolouch, CSc.
Konzultanti diplomové práce:

doc. Ing. Tomáš Kratochvíl, Ph.D.
Předseda oborové rady

UPOZORNĚNÍ:

Autor diplomové práce nesmí při vytváření diplomové práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

ABSTRAKT

Tato diplomová práce se zabývá řídícími jednotkami a jejich testováním. Pro tyto potřeby je zde popsána aplikace, která je schopná tyto řídící jednotky autonomně testovat. Aplikace má dvě části, první je generátor a editor dat a druhou je testovací procedura, která otestuje vytvořený stavový automat a vygeneruje záznam o provedeném testu.

KLÍČOVÁ SLOVA

Stavový automat, embedded zařízení, jazyk XML, MDI aplikace, stavový diagram, C#.NET, testovací procedura, vlákna

ABSTRACT

The subject of this master thesis are state machines and their testing. To this purpose, an application is described, which is able to test these state machines autonomously. Application have two parts, first generator and editor of data is built and second part is testing procedure, which is able to test state machines in question and to give corresponding test report.

KEYWORDS

State machine, embedded device, XML language, MDI application, state diagram, C#.NET, test procedure, threads

HORKÝ, Stanislav *Autonomní generátor testovacích skriptů*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav radioelektroniky, 2014. 75 s. Vedoucí práce byl prof. Ing. Jaromír Kolouch, CSc.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Autonomní generátor testovacích skriptů" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

(podpis autora)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu doc. Ing. Jaromíru Kolouchovi, CSc. a konzultantovi panu Ing. Jiřímu Macháčkovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

(podpis autora)

OBSAH

Úvod **10**

1 Teoretická část **11**

1.1 Systémový náhled 11
1.2 Řidící stavový automat 11
1.3 Formát XML 13

2 Aplikace **14**

2.1 Koncept 14
2.2 Základní popis 14
2.3 Vrstvy aplikace 16
2.3.1 Datové jádro 17
2.3.2 Přístupová vrstva 19
2.3.3 Přístup ke knihovnám I/O modulů 20
2.3.4 Testovací procedura 20
2.3.5 Hlavní formulář 20
2.3.6 Přidružené formuláře 21
2.4 Zpracování dat z programu MS Excel 22
2.4.1 Popis převodu 24
2.5 Grafický editor 26
2.5.1 Vykreslení jednotlivých objektů 27
2.5.2 Postup vykreslení při převodu ze stavové tabulky 32
2.5.3 Ostatní ovládací prvky editoru 33
2.6 Testovací prostředí 34
2.6.1 Základní princip testování 37
2.6.2 Konfigurace testovací procedury 40
2.6.3 Struktura testovací procedury 44
2.6.4 Vlákno procedury 47
2.6.5 Zapisování a prezentace výsledků 50
2.6.6 Modifikace aplikace 52

3 Test aplikace **54**

3.1 STK500 s procesorem ATmega16 54
3.1.1 Vytvořený projekt 54
3.1.2 Výsledky testů 56
3.2 Smart Valve SV9501 58
3.2.1 Vytvořený projekt 61

3.2.2 Výsledky testů 66

4 Závěr 69

Literatura 71

Seznam symbolů, veličin a zkratek 72

Seznam příloh 73

A Stavový automat 74

SEZNAM OBRÁZKŮ

1.1 Přehled systému 12
1.2 Obecná struktura stavové tabulky 13
2.1 Vrstvy aplikace 17
2.2 Struktura XML dokumentu pro vytvoření struktury v datovém jádře 18
2.3 Vývojový diagram grafického editoru 22
2.4 Zobrazení tabulky po načtení xls souboru 23
2.5 Zobrazení tabulky po vybrání koordinátů 25
2.6 Pracovní plocha uživatele 28
2.7 Okno pro vytvoření stavu 29
2.8 Vytvořené stavy 30
2.9 Vykreslování přechodů 31
2.10 Reálné vykreslení přechodů 32
2.11 Rozložení stavů při automatickém vykreslování 33
2.12 Příklady označování 34
2.13 Testovací prostředí 35
2.14 Ovládací menu testovacího prostředí 36
2.15 Stavový diagram a přechody mezi stavy 38
2.16 Postup konfigurace 41
2.17 Vložení metody 42
2.18 Nastavení vstupu 43
2.19 Editor testu 44
2.20 Struktura testování 45
2.21 Grafická prezentace průběhu testu 46
2.22 Vývojový diagram průběhu testování 49
2.23 Hlavička záznamu testu 51
2.24 Záznamu testu 52
3.1 Diagram řídícího automatu pro zařízení s vývojovou deskou STK500 . 55
3.2 Fotografie pracoviště s vývojovou deskou STK500 56
3.3 Výsledek prvního testu stavového automatu zařízení STK500 57
3.4 Výsledek druhého testu stavového automatu zařízení STK500 58
3.5 Schéma zapojení z instalačních instrukcí[9] 59
3.6 Fotografie pracoviště s SV9501 60
3.7 Stavový diagram z instalačních instrukcí[9] 62
3.8 Diagram řídícího automatu zařízení SV9501 64
3.9 Diagram řídícího automatu zařízení SV9501 pro cyklický test 65
3.10 Výsledky prvního testu stavového automatu zařízení SV9501 67
3.11 Výsledky druhého testu stavového automatu zařízení SV9501 68

A.1 Stavová tabulka automatu 74
A.2 Postup vytváření struktury v jádře 75

ÚVOD

Tato práce je zaměřena na automatizaci funkčního softwarového testování řídících jednotek

Cílem této diplomové práce je vytvořit aplikaci, která bude obsahovat grafický editor a jádro s testovacím algoritmem. V grafickém editoru bude možné vytvářet stavové automaty, buď na základě stavové tabulky, nebo samotným tvořením stavového diagramu. Uživatel by měl mít možnost vytvořit komplexní popis řídící jednotky, které bude připojeno pomocí vstupně/výstupních modulů k počítači. Testovací prostředí bude schopné otestovat stavový automat zařízení a informovat uživatele o průběhu a výsledcích testu v grafické i textové podobě.

Práce je členěna do čtyř částí. První kapitola obsahuje teoretický úvod do problematiky zadání. Druhá kapitola popisuje samotnou aplikaci, její koncept a celkovou strukturu. Třetí kapitola obsahuje test kompletní aplikace a výsledky tohoto testu a poslední kapitola představuje stručné shrnutí celého dokumentu.

1 TEORETICKÁ ČÁST

1.1 Systémový náhled

Na obr.1.1 je nastíněný obecný pohled na celý systém. Na tomto základě je postavena celá práce. Prakticky se jedná o ověření, že je možné pomocí takovéhoto systému lze testovat jakékoliv zařízení. Celý systém se skládá z řídící jednotky - osobního počítače nebo laptopu, které uživatel bude využívat k vytvoření programu, který bude schopný autonomně testovat zvolené zařízení. Řídící element bude možné připojit k testovanému zařízení za pomocí dalších jednotek. Na obrázku jsou uvedené příklady jako sériová linka, USB kabel nebo vstupy a výstupy I/O karty, která může umožňovat digitální a analogové hodnoty signálů. Tento systém vychází z faktu, že k samotnému počítači nelze připojit testované zařízení přímo, z důvodů široké variability vstupů a výstupů testovaného zařízení.

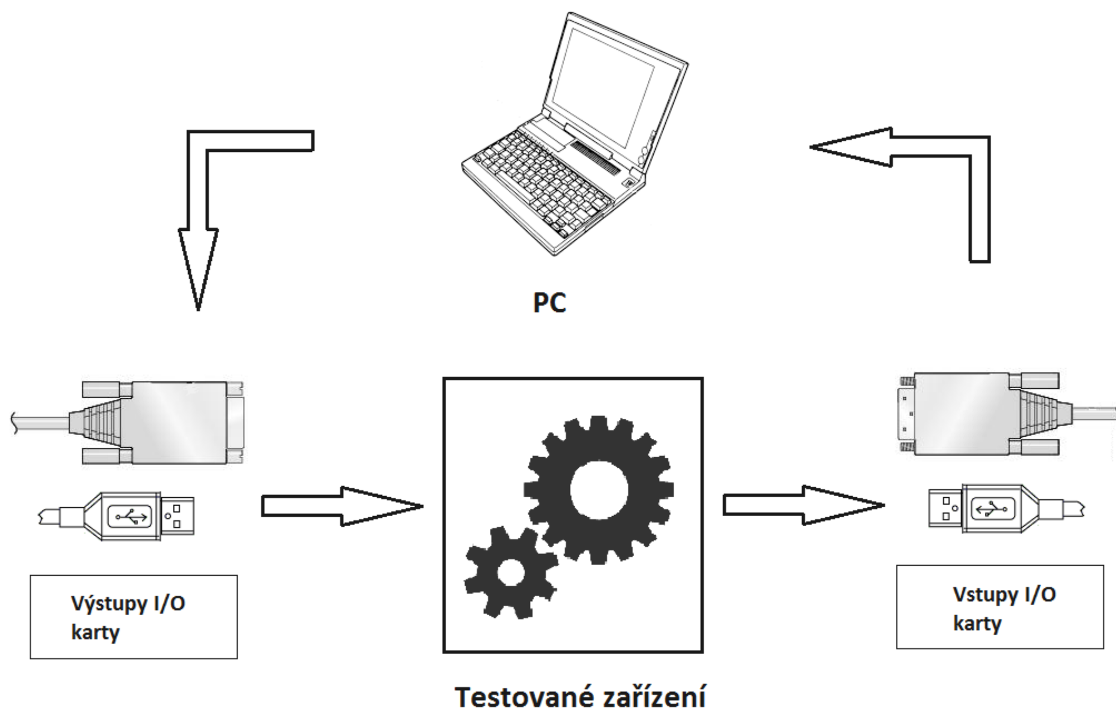
Testování bude probíhat na základě řídících impulzů od PC, které budou dodávány pomocí externích jednotek do testovaného zařízení a odezvy zařízení budou zpracovány dalšími externími jednotkami zpět do PC. Aplikace v PC následně vyhodnocuje impulzy od zařízení a autonomně se rozhoduje jaké výstupní impulzy budou vyslány směrem k testovanému zařízení. Jedná se tedy o uzavřený kruh, kdy je vše řízeno pomocí PC. Do bloku testovaného zařízení je nutné si dosadit kompletní zařízení i s nutnými podpůrnými moduly jako napájecí napětí, potřebné zátěže a podobně.

V této práci bude hlavním cílem aplikace, která bude schopná takovýto autonomní systém realizovat. Další externí zařízení potřebná k vytvoření takovéhoto systému nebudou analyzována.

1.2 Řidící stavový automat

Embedded systém je jednoúčelový systém, který je součástí většího elektrického a mechanického celku. Nejčastěji je omezený na výpočty v reálném čase a jeho softwarové testování probíhá velice pečlivě, protože zařízení musí splňovat bezpečnostní požadavky, které kladou důraz na bezpečnost lidí a jejich majetek. [5]

Nedílnou součástí takovéhoto systému je řidící stavový automat a tedy i jeho testování. Automat můžeme popisovat stavovou tabulkou, stavovým diagramem, stromovou strukturou, nebo také přímo elektrickým schématem. Pro přehledné znázornění funkce stavového automatu je nejvýhodnější použít stavový diagram nebo stromovou strukturu. Díky tomu je možné vytvářet řidící stavové automaty s takovou strukturou, která je vyžadována.



Obr. 1.1: Přehled systému

Stromová struktura je abstraktní datový typ, který má charakter stromu, tedy jakýsi kořen celé struktury a z něj vycházející větve, které jsou navzájem propojené uzly.[6]

Pokud je v práci zmíněná stavová tabulka, jedná se o tabulku, ve které jsou logicky a přehledně umístěny veškeré struktury, které popisují řídící stavový automat. Struktura takovéto tabulky je na obr.1.2, není ale žádná unifikovaná metoda na vytváření takovýchto tabulek, pouze v případě této práce se bude uvažovat o tabulce v uvedeném formátu.

K základním seznamům musí být navíc pro identifikaci doplněny zkratky a v případě samotných stavů i jejich číslování.

Stavový diagram je diagram používaný v počítačové technice a příbuzných odvětvích k popisu chování různých systémů. Předpokládá se konečný počet stavů, mezi kterými se přechází pomocí událostí.[8]

Příkladem může být tabulka pro řídící stavový automat na obr.A.1, která byla vytvořena v programu MS Excel. Tabulka v příloze splňuje strukturu, která je definována na obr.1.2. Automat obsahuje deset stavů a pět událostí. Kontrolovány jsou hodnoty A/D převodníku a časovače. Takováto struktura se objevuje v drtivé většině zařízení. Vytvořená stavová tabulka nepopisuje reálné používané zařízení, je zde uvedena pouze jako příklad.

	Události	Výstupy	Proměnné
Stavy	Následující stav	Výstup aktivní [boolean]	Hodnota proměnné

Obr. 1.2: Obecná struktura stavové tabulky

Vytvořená aplikace bude schopná pracovat s řidícím stavovým automatem, popsaném tabulkou na obr.1.2, která má předem definovanou strukturu. Obsahuje seznam stavů, událostí, výstupů a proměnných. Ke každému stavu může být přiřazena událost (neboli změna), která nese informaci o přesunu do dalšího stavu. Dále může být ke stavu přiřazen aktivní výstup a proměnné s určitými hodnotami, které ovlivňují přechody.

1.3 Formát XML

Jednou z přehlednějších struktur pro popsání a zobrazení řidící stavového automatu je stromová struktura, pro jejíž popis může být použit jazyk XML.

Extensible Markup Language, zkráceně XML, popisuje třídu datových objektů zvaných XML dokumenty a popisuje počítačové programy, které tyto dokumenty zpracovávají. XML je omezená forma SGML, Standard Generalized Markup Language [ISO8879].[1]

XML dokumenty jsou vytvořeny z entit, které obsahují analyzovaná nebo neanalyzovaná data. Analyzovaná data jsou tvořena znaky, z nichž některé jsou formou dat a některé formou značek. Značky popisují vzhled dokumentu a jeho logickou strukturu. XML umožňuje vytvářet přesně definované dokumenty a logické struktury.[1]

2 APLIKACE

2.1 Koncept

Kompletní aplikace bude mít dvě hlavní části - generátor/editor dat a spouštěč testovacího algoritmu. Uživateli by mělo být umožněno vytvářet stavové diagramy přímo, bez jakýchkoli vstupních souborů, jako prázdný projekt. Další možností bude vytvořit automat ze stavové tabulky - ze souboru xls. Grafický editor musí umožnit uživateli používat všechny editační možnosti - přidání, editace a mazání objektů.

Aplikace se bude skládat z více vrstev. Takováto struktura znamená, že zobrazování, zpracování a vyhodnocování dat je od sebe odděleno. Tímto oddělením je vývojáři umožněno modifikovat jednotlivé části, nebo je případně využít v jiné aplikaci. [7]

Jádrem programu bude stromová struktura, která bude uložena na disku ve formátu XML a bude obsahovat veškeré informace potřebné pro další zpracovávání dat. Soubor XML bude uložen samostatně, nebo jako celý projekt (XML soubor i s daty z grafické části). Pro potřeby testování stavového automatu bude následně potřeba specifikovat akce které bude muset aplikace provádět, například zjištění hodnoty určitého výstupu, případně nastavení hodnoty, nebo přečtení hodnoty časovače a jeho porovnání.

Výstupem z testovacího algoritmu bude určitý typ zprávy, která ponese informaci zda stavový automat zařízení proběhl správným způsobem a když ne, tak v jaké fázi nastala chyba. Pro vizualizaci testování bude použito již vytvořeného stavového diagramu, pouze v testovací části nebude možná jeho editace.

2.2 Základní popis

Vývojový diagram grafického editoru je uveden na obr.2.3. Po výběru nového projektu je nutné zadat jméno nebo identifikační číslo osoby, která projekt vytváří a navíc je nutné zadat jméno zařízení, jehož stavový automat budeme vytvářet. Datum a čas vytvoření projektu automaticky doplňuje aplikace. Pokud nebude vyplněno ani jedno pole, automaticky se doplní do obou polí text "Unknown". Datum a čas doplňuje aplikace, aby bylo možné dohledat projekt alespoň podle těchto údajů.

Když je vybrána možnost "Blank project", uživatel aplikace chce vytvářet stavový automat bez jakýchkoliv vstupních dat. Struktura, která tvoří jádro programu a která bude popsána v další kapitole se vytvoří pouze s identifikátory a časem, kdy byl projekt vytvořen.

Pokud budeme vytvářet projekt ze souboru xls, je nutné nejprve mít takovýto soubor vytvořený a to dle vzoru, který je uveden v kapitole 1.2. Aplikace bude umožňovat editaci stavové tabulky, ale tyto úpravy se následně neodrazí v xls souboru. Bylo rozhodnuto, že pokud je editace možná v xls souboru, uživatel by měl upravovat xls soubor. Podrobnosti budou popsány v kapitole 2.4. Protože každý soubor xls obsahuje více pracovních ploch, uživatel vybere takovou plochu, která obsahuje stavovou tabulku. Následně je do struktury dataGridView zobrazena celá pracovní plocha ze souboru xls.

Každá stavová tabulka může být vytvořena různými způsoby. Převedení tabulky na stromovou strukturu je možné v základu rozdělit na tři části. První je plně automatizovaný převod. V tomto případě by uživatel pouze vybral první pole tabulky a aplikace by provedla vše ostatní. Není ale možné obsáhnout úplně všechny možnosti a vytvoření takového algoritmu by zabralo velice mnoho času a navíc by nebylo ve výsledku zaručeno, zda by algoritmus reagoval správně. Dalším extrémem je čistě manuální převedení - uživatel by musel definovat kde a kolik je všech stavů, kde jsou jejich zkratky, který přechod patří ke které události atd. Při velkém objemu dat, které budou tabulky s největší pravděpodobností obsahovat by tato práce byla zdlouhavá a nebylo by možné vyloučit chybu lidského faktoru.

Proto byl vybrán třetí způsob - poloautomatický převod. Uživatel bude muset definovat určité body, které aplikaci řeknou kde má startovní hodnoty a zbytek je už automatický. V tomto případě se nevyhneme určité unifikaci stavových tabulek, tak jak bylo popsáno v kapitole 1.2. V tomto případě je nutné pouze dodržet rozložení jednotlivých struktur a hlavně doplnit veškeré identifikátory. Podrobnější postup převodu bude vysvětlen níže.

Aby se zamezilo chybám ve stavovém automatu, je nutné kontrolovat zda tabulka obsahuje správné formáty dat, neobsahuje stejné zkratky a tedy i názvy. V takovémto případě by došlo k narušení správné funkce stavového automatu. Proto se v této fázi provádí kontrola dat a je umožněno aby uživatel mohl případné chyby odstranit. Pokud je kontrola dat v pořádku, vytvoří se stromová struktura stavového automatu. Ihned poté se struktura převede do formátu XML a uloží se jako dočasný soubor. Po této akci je ze souboru vytvořen stavový diagram a uživatel může další úpravy provádět přímo v grafickém editoru. Výsledně uložený projekt by měl následně obsahovat stavový diagram, který popisuje stavový automat zařízení dle uživatelových představ.

Po přepnutí z grafického editoru do testovacího prostředí dojde pouze ke změně ovládacích prvků, kdy je uživateli umožněno spouštět a přerušovat test. Samotné zpracování testu je zjednodušeně naznačeno v příloze na obr.2.22. Podrobnější informace jsou uvedené v kapitole 2.6. Testování je prováděno v odděleném vláknu než ve kterém je zpracováván zbytek aplikace. Jedním z důvodů, kvůli kterému bylo

zvoleno nové vlákno je fakt, že v testování mohou vzniknout dlouhé mezery kdy se např. čeká na odezvu od zařízení a pokud by v tuto chvíli uživatel minimalizoval aplikaci, nebylo by možné všechna okna znovu vykreslit, nehledě na fakt že by nebylo možné test přerušit, protože by hlavní formulář nedokázal ošetřovat události, které nastanou od tlačítek a ovládacích prvků.

Průběh testovacího vlákna je jedna nekonečná smyčka, která má za úkol prověřit všechny události, které mohou ve stavovém automatu nastat. Uživatel může testování přerušit, v tomto případě nebude vytvářena zpráva o úspěšnosti testování. Dalšími možnostmi je že test bude úspěšně dokončen, nebo z nějakého důvodu nebude popis stavového automatu zařízení korespondovat se skutečným popisem. V tomto případě je generována zpráva o výsledku testu, aby bylo možné identifikovat zda zařízení odpovídá specifikaci, nebo zda je chyba v popisu stavového automatu, který vytvořil uživatel v testovací aplikaci.

Pro správný běh testovacího algoritmu je nutné nastavit určité množství informací, jejichž objem se zvyšuje se složitostí stavového automatu testovaného zařízení. V první řadě je nutné definovat testovací proceduře nástroje se kterými bude pracovat. Tímto je myšlen konfigurační soubor pro I/O kartu a funkce, které bude aplikace využívat pro ovládání zařízení přes zvolenou I/O kartu. Při samotném vytváření stavového automatu a jeho následné konfiguraci je nutné mít hlubší povědomí o funkci a principech takovýchto stavových automatů.

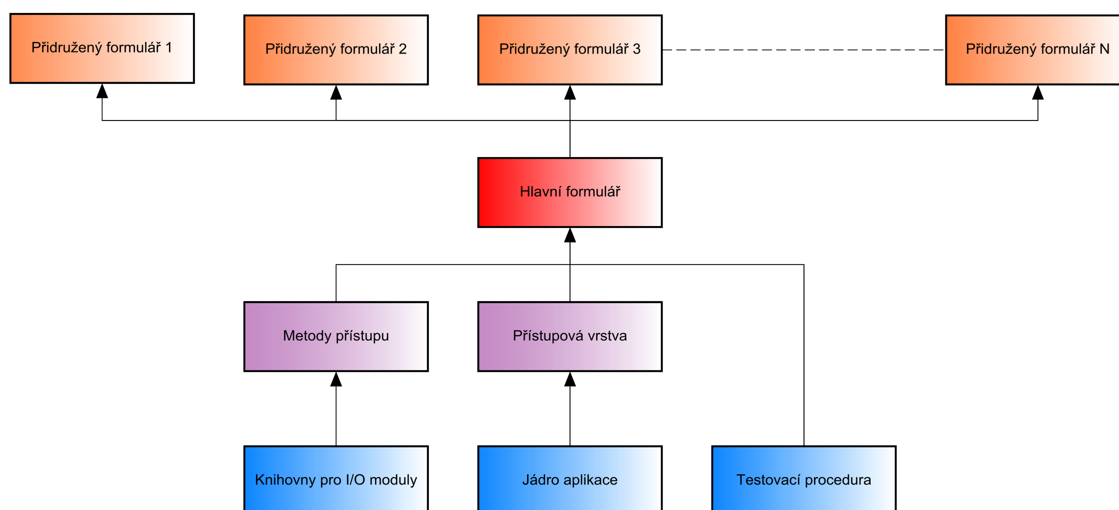
2.3 Vrstvy aplikace

Celá aplikace je koncipována jako MDI (Multiple Document Interface). To znamená aplikaci, která umožňuje zobrazovat více dokumentů ve stejný čas, z nichž každý má svoje vlastní okno. MDI aplikace má nejčastěji hlavní menu s různými ovládacími prvky, které umožňují přepínání mezi okny a dokumenty.[4]

Struktura je naznačena na obr.2.1. Každý blok je barevně označen, aby bylo naznačeno do které oblasti spadá jeho funkce. Nejvyšší oranžová vrstva slouží ke komunikaci a interakci s uživatelem. Jsou tím myšlena grafická rozhraní, menu a konfigurační okna. V této aplikaci bude pouze jedno hlavní grafické rozhraní, ale jak je naznačeno, počet oken, které takto lze vytvořit je prakticky neomezený. Všechna tato okna jsou sdružena pod hlavní formulář, označený červeně. Tento formulář obsahuje jak ovládací prvky celé aplikace, tak i algoritmy, kterými spojuje uživatelské rozhraní s ostatními nižšími vrstvami, zajišťuje komunikaci a přenos informací mezi jednotlivými vrstvami a také je všechny zpracovává a rozhoduje, co s nimi bude následně provedeno.

Modrou barvou jsou odlišeny nejnižší vrstvy aplikace. Jádro aplikace bude po-

drobněji rozebráno níže, stejně jako blok Testovací procedura. Knihovny pro I/O moduly jsou obecným blokem, který není přesněji definovaný a není součástí aplikace, pouze se k němu přistupuje. Obsahuje funkce, které obsluhují fyzické I/O moduly a umožňují základní ovládání a čtení informací z testovaného zařízení. Aby bylo zamezeno přímému přístupu k jádru aplikace a knihovnám pro I/O moduly, byla mezi hlavní formulář a nejnižší vrstvu aplikace vložena takzvaná přístupová vrstva označená fialovou barvou. Tato vrstva se stará o obsluhu struktur a funkcí na nejnižší vrstvě.

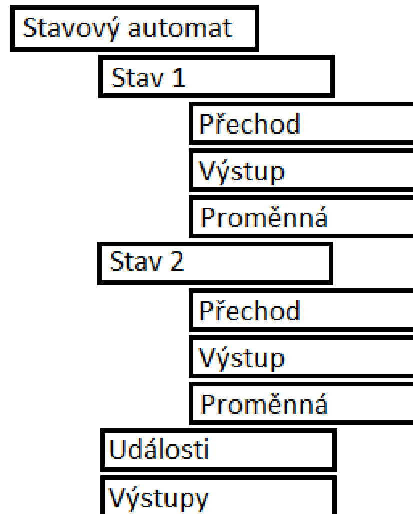


Obr. 2.1: Vrstvy aplikace

2.3.1 Datové jádro

Jádro aplikace tvoří stromová struktura. Stavová tabulka uvedená v předchozí kapitole 1.2 je pro zorientování a pochopení průběhu a funkce řidícího stavového automatu vhodná pro uživatele, ale nevhodná pro algoritmické zpracování. Pokud chceme použít pro popis stromovou strukturu v XML dokumentu, je třeba definovat dokument a jeho logickou strukturu. Taková logická struktura XML dokumentu je naznačena na obr.2.2. Kořenem samotné struktury je stavový automat, jehož vlastnostmi bude datum vytvoření dokumentu, identifikátor osoby a testovaného automatu. Stavový automat bude obsahovat seznam stavů a každý stav v sobě ponese informaci o tom která událost povede ke změně stavu, výstupu, který je aktivní a proměnné která se má sledovat. Dále bude dokument obsahovat seznam všech událostí a výstupů.

Řidící stavový automat je tímto způsobem popsán velice hrubě a pokud jej chceme testovat, musí se do XML dokumentu (stromové struktury) přiřadit mnohem



Obr. 2.2: Struktura XML dokumentu pro vytvoření struktury v datovém jádře

více informací, příkladem mohou být metody, které má aplikace volat aby nastavila výstup nebo přečetla vstup ze vstupně/výstupních modulů, které tvoří propojení mezi aplikací a zařízením. Jak ale bylo řečeno výše, XML jazyk toto bez problému umožňuje. Další jeho výhodou je univerzálnost zpracování a v jazyku C# je umožněno pracovat s XML dokumenty velice jednoduše.

Tato stromová struktura v datovém jádře byla vytvořena pomocí datového objektu *Dictionary<>*, která je k dispozici v jazyku C#. Bylo třeba se rozhodovat mezi strukturou *List<>* a *Dictionary<>*, protože ale potřebujeme přesně vědět, kam se chceme ve struktuře odkazovat, je výhodnější použít *Dictionary<>*, protože se do něj data neodkazují pouze číslem, ale identifikují se podle jakékoliv proměnné kterou programovací jazyk podporuje. V případě této aplikace se ve slovnících indexuje pomocí proměnné typu *string*. Pokud totiž například vytvoříme slovník s parametry *Dictionary<string,Tstate>*, bude se do tohoto slovníku ukládat definovaná struktura *Tstate* (která obsahuje další informace) pod klíčem ve formátu *string* a tímto klíčem bude v tomto případě zkratka stavu. Tudíž stačí znát zkratku a podle ní lze ve struktuře dohledat proměnnou která je vyžadována.

Kořenem je třída *StateMachine*, která obsahuje tři slovníky - *statesDictionary*, *eventsDictionary* a *outputsDictionary*, tedy seznam všech stavů, událostí a výstupů. Třída *StateMachine* má jako další proměnné jméno uživatele, název testované jednotky, datum a čas vytvoření a cestu ke složce ve které je uložen XML soubor. Cesta ke složce se vyplňuje až v případě, že se má struktura uložit. Také identifikuje, zda se soubor modifikoval.

Slovníky událostí a výstupů obsahují další struktury, které pouze definují přesný název a dále metodu, kterou bude nutné zavolat, pokud se bude stavový automat

testovat.

Každá struktura *Tstate* ve slovníku stavů obsahuje další tři slovníky - *transitToNextDict*, *outputValueDict* a *variableValueDict*. První slovník obsahuje informaci o přechodu do dalšího stavu, druhý slovník obsahuje aktivní výstupy a poslední slovník hodnoty proměnných.

Slovník *eventsDictionary* obsahuje všechny události které mohou ve stavovém automatu nastat. Pod tímto pojmem si lze představit např. sepnutí tlačítka, rozepnutí tlačítka, nebo vypršení hodnoty časovače. Poslední slovník *outputsDictionary* obsahuje veškeré výstupy ze zařízení, které uživatel uzná za vhodné využívat. Ve slovníku jsou obsaženy identifikátory, které umožňují číst hodnoty těchto výstupů (pro aplikaci jsou to vstupy).

Ve zdrojovém souboru jádra (StateMachine.cs) jsou dále obsaženy funkce, které strukturu naplňují. Přesné vytvoření struktury je ale provedeno pomocí vyšší vrstvy.

Jádro programu má tedy jediný úkol - obsahuje v sobě veškeré informace, které jsou potřeba pro vytvoření stavového automatu.

V průběhu práce se struktura datového jádra měnila, jelikož bylo nutné reagovat na požadavky jak grafického editoru, tak samotné testovací procedury. Zde v této kapitole je popsáno podrobně, z jakých objektů se jednotlivé části datového jádra skládají. Objektů *Dictionary*<> bylo využito ve velkém množství, protože struktura se dělí na další struktury, které je třeba indexovat jakoukoliv proměnnou, nejenom číslem a tím je možné do struktury zadat více informací o určitém objektu, které se následně mohou intuitivně vyhledávat pomocí proměnné a ne je složitěji vyhledávat pomocí číselného pořadí, které umožňuje struktura *List*<>.

2.3.2 Přístupová vrstva

Jak již bylo řečeno výše, přístupová vrstva se stará o přístup k jádru. Dále poskytuje pro určité funkce další stupeň kontroly, který se neprovádí na vyšší vrstvě. Mezi hlavní úkoly přístupové vrstvy patří vedle samotného sestavení struktury také vytváření a ukládání struktury do formátu XML.

Mezi další funkce patří získávání základních informací o struktuře jako např. datum a čas, jméno uživatele a počet objektů ve slovnících. Tyto informace jsou dále předávány vyšší vrstvě. Počet objektů ve slovnících je důležitý pro orientaci ve stavové tabulce.

Poslední část přístupové vrstvy tvoří funkce které pracují již s objekty ve struktuře, ne s celou strukturou. Mohou to být např. funkce které přidají, odeberou nebo získají informace o určitém stavu, výstupu, události nebo proměnné.

2.3.3 Přístup ke knihovnám I/O modulů

V tomto bloku je umožněn přístup ke knihovnám. Základní funkcí je předložit uživateli potřebné funkce a akce, které může použít a získat o nich potřebné informace aby bylo možné tyto funkce automaticky volat. Dále jsou zde obsaženy pomocné funkce které přímo zavolají určitou funkci, nakonfigurují knihovny a spustí potřebné konstruktory. Mezi doplňkové funkce patří také kontrola vstupních proměnných z testovaného zařízení a resetování zařízení.

Pokud by uživatel změnil knihovnu, je třeba zkontrolovat, zda dostupné přístupové funkce jsou dostačující. Samotná aplikace by měla být teoreticky nezávislá na použité knihovně, v praxi je ale třeba zkontrolovat jak přesně jsou funkce volané a co všechno knihovna obsahuje. Problémem může být použití knihovny, která se svojí strukturou může velice lišit od použité testovací procedury a způsobu kterým aplikace funguje.

Knihovna použitá v této práci je majetkem firmy Honeywell a tento majetek je k dispozici přes NDA. Z tohoto důvodu není funkčnost a struktura této knihovny v práci popisována.

2.3.4 Testovací procedura

Tato vrstva obsahuje strukturu do které jsou ukládány výsledky testu. Celková funkčnost vrstvy bude popsána níže. V této fázi lze říct, že vrstva funguje jako zdroj dat pro rozhodování, které události již proběhly a které bude nutno dále zavolat a také obsahuje identifikátory v kterém stavu se zařízení nachází a v jakém stavu se nacházelo. Tato vrstva je vytvářena až ve fázi, kdy se spustí testovací procedura.

2.3.5 Hlavní formulář

Jako hlavní formulář je zde myšleno hlavní okno MDI aplikace, do kterého se zobrazují ostatní okna, která slouží ke komunikaci s uživatelem, která v aplikaci dále tvoří další vrstvu a hlavní okno je pouze sdružuje do jednoho celku. Tato okna jsou tvořena objektem Windows Form, což je základní jednotka aplikace, která umožňuje vývojáři naplnit ji ovládacími prvky a zobrazovacími okny, které umožňují uživateli s aplikací pracovat.[3]

Tato vrstva slouží k propojení jednotlivých oken obsažených v aplikaci. Základem je hlavní okno (jako například v aplikaci MS Word), ve kterém jsou zobrazována a sdružována všechna ostatní okna. Systém MDI aplikace tedy používá více otevřených oken a je potřeba změnu v jednom převést do druhého. Příkladem může být editace stavového diagramu, kdy je potřeba vytvořit nový stav. Samotný editor má vlastní okno, které slouží k vizualizaci dat, ale samotná editace nebo vytváření

struktur vyžaduje složitější ovládání, respektive více ovládacích prvků, které bude vysvětleno níže. Pokud by bylo vše v jednom okně, pro samotnou vizualizaci by zbývalo málo místa, nehledě na to že velikost zobrazení se také odvíjí od velikosti monitoru uživatele.

Z tohoto důvodu je vytvořené okno, které sdružuje veškerá další okna do jednoho celku. Stará se také o přístup do přístupové vrstvy a obsahuje základní příkazy jako vytvoření a uložení jednotlivých částí projektu, ovládání testu, přepínání prostředí a také editační příkazy. Dále je možné tuto vrstvu použít pro další stupeň kontroly funkcí, které jsou určené spíše pro přidružené formuláře a tudíž jsou odděleny od přístupové vrstvy.

Úkolem této vrstvy je tedy přenášet data mezi přidruženými formuláři a přístupovou vrstvou k jádru, mezi bloky které jsou použity pro testování, ale také mezi samotnými přidruženými formuláři. Byla testována i možnost předávat přímo mezi jednotlivými okny, ale v konečném důsledku nebylo možné provést to u natolik složité aplikace.

Tento vzhled aplikace byl také zvolen kvůli podobnosti s Windows aplikacemi jako MS Word a Excel, nebo aplikace jako Visual Studio, a to z toho důvodu že tyto styly aplikací jsou pro uživatele již známé a neměl by být velký problém se v ovládacích prvcích takovéto aplikace zorientovat.

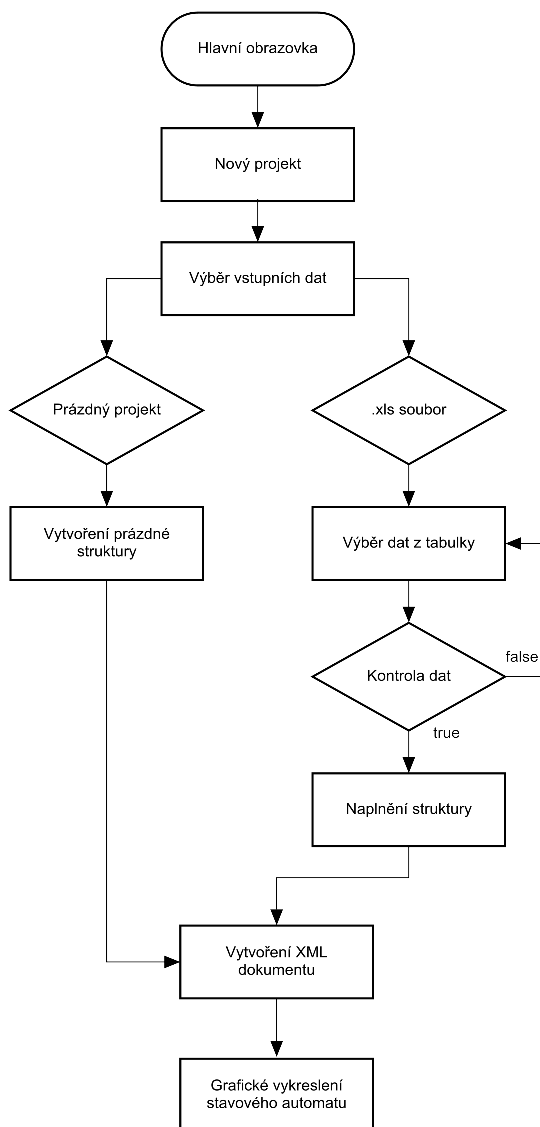
2.3.6 Přidružené formuláře

Okna, která se zobrazují nad hlavním formulářem slouží k vizualizaci dat pro uživatele a provádějí jej celým procesem vytváření projektu stavového automatu. Obsahují všechny ovládací prvky které je nutné používat a také pokrývají chyby, kterých se může uživatel dopustit a upozorňují jej vyskakovacími okny. Nižší vrstvy jsou ošetřeny bez vyskakovacích oken, pouze posílají informaci o špatně provedené akci, která je následně výše zpracovávána a upozorní uživatele znovu vyskakovacím oknem.

Protože aplikace musí být schopná vytvářet stavové automaty, musela tato vrstva být vytvořena natolik univerzálně, aby bylo umožněno uživateli vytvořit a otestovat stavový automat přesně podle jeho představ. Celý projekt obsahuje velké množství informací, které je nutné ručně vytvořit, ale není možné obsáhnout veškeré možnosti, tedy aplikace je úzce zaměřená na koncového uživatele, který musí mít určité zkušenosti a povědomí o stavových automatech a jejich funkci.

V konečném důsledku tato vrstva odráží vývojový diagram uvedený na obr.2.3. Tento diagram popisuje vytvoření grafického rozhraní, ale nereflektuje vytvoření testovací procedury a samotné testování. Tento postup bude vysvětlen dále. Největší důraz byl kladen na samotnou grafickou část aplikace.

Úkolem této vrstvy je tedy komunikovat s uživatelem a umožnit mu modifikovat a upravovat celý projekt.



Obr. 2.3: Vývojový diagram grafického editoru

2.4 Zpracování dat z programu MS Excel

V této kapitole bude přiblížen postup převodu stavové tabulky na diagram. Po načtení xls souboru se zobrazí okno, které je na obr.2.4. V levém horním rohu označeném zeleným rámečkem (tyto barevné rámečky jsou zde pouze pro odlišení, v samotné aplikaci se nevyskytují) jsou informace získané z právě vytvořené struktury automatu - jméno uživatele, název testované jednotky, datum a čas. Před nahráním

xls souboru je uživatel samozřejmě nucen tato pole vyplnit. Celý postup aplikací bude popsán dále.

V červeném rámečku jsou pole pro určení koordinátů. Jak bylo zmíněno v kapitole 2.2, převod tabulky na stromovou strukturu je poloautomatický. Je tedy třeba zvolit začáteční body stavů, událostí, výstupů a proměnných. Pokud uživatel nezvolí všechny souřadnice, aplikace nedovolí pokračovat, dokud nebudou vyplněny.

Ve žlutém rámečku jsou pak uvedeny chyby, které byly zjištěny v tabulce. Jedním z problémů je to, že uživatel může zvolit různé pozice a vytvořit si tak celou strukturu nesprávně. Tento případ ale není možné kontrolovat, protože aplikace nemůže přesně vědět jaký stavový automat chce uživatel vytvořit. Částečně byl tento problém podchycen kontrolou chyb. Mezi sledované parametry patří zda není zadávána stejná zkratka (ve všech případech jak u stavů, výstupů a proměnných), kdyby tato chyba nastala aplikace by se ukončila, protože slovníky obsažené ve struktuře neumožňují přidání stejného klíče. Jelikož u výstupů je přesně definovaná hodnota, kterou může v tabulce nabývat (logická jednička), je tato hodnota také kontrolována. Tímto je zajištěno alespoň to, že pokud uživatel nesprávně nastaví souřadnice výstupu, je na tuto chybu upozorněn.

A	B	C	D	E	F	G	H	I	J
Error Example									
			Events			Outputs		Variables	
number	symbol	state	Internal Error	Test Mode Eneterd	Timer Expires	Valve	A/D	Timer	Threshold
			IER	IER	FL	VL	AD	TM	TM
0	IDL	Idle	SLO	IDL	SLO		JL	1	0,2
1	ST1	State 1	SLO	IDL	ST2		1	12	0,2
2	ST1	State 2	SLO	IDL	SLO		1	5,2	1,5
3	ST3	State 3	SLO	IDL	RUN	12	1	1,8	0,4
4	ST3	Run	SLO	IDL	ST4	1	1,2	1,5	0,4
5	ST4	State 4	SLO		ST1		1	10	
6	SLO	Soft Lockout					kldsjf	300	
7									
8									

Obr. 2.4: Zobrazení tabulky po načtení xls souboru

Na obr.2.4 si lze povšimnout, že do tabulky jsou záměrně vložené chyby - některé stavy se opakují, ve výstupech jsou nesprávné hodnoty a je číselně více stavů, ale

nevyplněných, tabulka může obsahovat dodatečné informace a aplikace musí podchycovat všechny, které by mohly zapříčinit nesprávné vytvoření celé struktury. Na obr.2.5 lze vidět co se stane, pokud se uživatel pokusí takovouto tabulku převést na strukturu v jádru aplikace. Pro přehlednost je buňka v tabulce, která obsahuje chybu zvýrazněna červeně. Zobrazování informací o chybách se jedná slouží okno nejníže označené jako Errors. Formát chyby je takový, že v hranatých závorkách jsou souřadnice buňky a následuje identifikace chyby a způsob jak se jí vyvarovat.

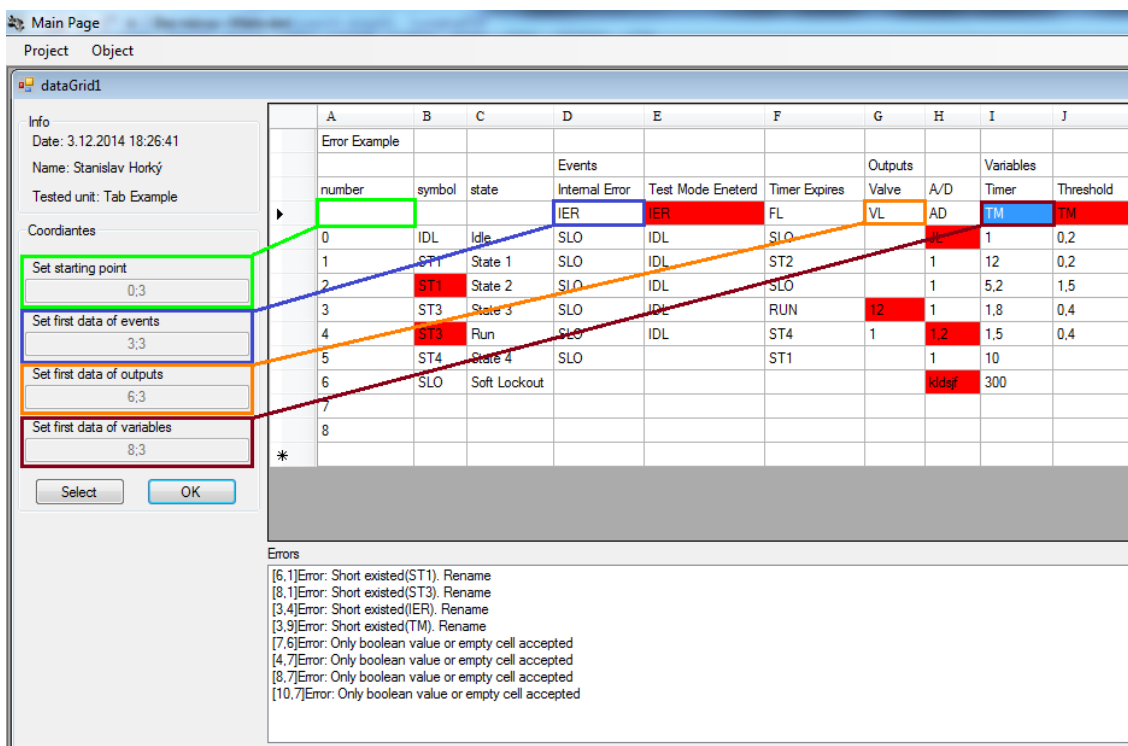
Aplikace uživateli nedovolí pokračovat dále, dokud nebudou všechny chyby odstraněny. Jak již bylo řečeno výše, je možné tabulku v aplikaci modifikovat, ale tyto změny se nepřevádějí zpětně do xls souboru. Takovýto způsob modifikace by de facto odstranil funkci aplikace MS Excel a ta by sloužila pouze jako jakési ukládací médium. V navržené aplikaci je už ukládací médium řešeno pomocí XML dokumentu, který je jednodušší pro algoritimické zpracování ve stromové struktuře než stavová tabulka v xls souboru.

Dále jsou na obr.2.5 vyznačeny souřadnice, kam je musí uživatel umístit, aby se převod na strukturu do jádra aplikace provedl správně. Nutné je vybrat první sloupce, kde začínají stavy, výstupy, proměnné a události. V případě stavů se musí zvolit první sloupec v pořadí *číslo stavu, zkratka stavu, název stavu* (tedy sloupec s čísly stavů), protože aplikace ukládá tyto informace přesně v pořadí v jakém jsou zde uvedené. Pozice řádků se zase musí zvolit tak, aby byly o jednu pozici *nad* samotnými informacemi o tom, co se bude v jakém stavu dít (tedy respektive musí být řádek nad prvním stavem v tabulce).

2.4.1 Popis převodu

Pokud je tabulka v pořádku, provede se její převod do struktury v jádře aplikace. V přístupové vrstvě jsou definovány čtyři dočasné slovníky, jeden pro stavy, druhý pro výstupy, třetí pro události a čtvrtý pro proměnné. Dále je zde definována tabulka dat.

Dle výběru koordinátů, které byly zmíněné výše, se do těchto struktur ukládají veškeré informace o elementech stavové tabulky a informace o nich. Každý element musí obsahovat svoji zkratku, pod kterou se do dočasných slovníků ukládá. V případě stavů se ukládá také číslo stavu. Veškeré informace jsou ve formě textu, takže například stav Idle s číslem 0 a zkratkou IDL se do slovníku uloží pod klíčem "IDL" a textem "0.Idle". Toto se provede pro všechny stavy dle koordinátů a stejně tak se provede pro ostatní elementy. Kontroluje se konec tabulky, tedy pokud algoritmus narazí na prázdné pole v tabulce při vytváření slovníků, znamená to konec tabulky a přejde se na další slovník.



Obr. 2.5: Zobrazení tabulky po vybrání koordinátů

Jakmile se provede převedení elementů tabulky do slovníků, spočítá se počet položek ve vytvořených slovnících. Tato čísla udají rozměry pro tabulku dat. Vertikální velikost určuje počet elementů ve slovníku stavů, horizontální rozměr určuje součet elementů v ostatních slovnících. Jakmile je tento krok provedený, do vytvořeného pole o rozměrech X a Y se převedou zbylá data z tabulky - to jsou přechody, hodnoty výstupů a hodnoty proměnných. Všechno je v textovém formátu. V tomto kroku již není potřeba dále udržovat v paměti aplikace tabulku stavového automatu, protože veškeré informace které potřebujeme pro vytvoření stavového diagramu máme v přístupové vrstvě k jádru.

Po této akci se provádí samotné vytváření struktury v jádře. Základní kořen *StateMachine* je již vytvořený na začátku nového projektu, stejně jako prázdné slovníky stavů, událostí a výstupů. Postup vytváření struktury je na obr.A.2. Postupuje se přes slovník stavů. Každý stav se vytvoří v jádře ve slovníku stavů s jediným rozdílem, že se do struktury samotného stavu v jádře ukládá odděleně název stavu a jeho číslo, již ve formátu čísla a ne ve formátu textu. Kontroluje se, zda stav již existuje, v tomto případě je to druhá kontrola, která by již měla být odstraněna ve vyšší vrstvě a pokud se podařilo tuto kontrolu nějakým způsobem obejít, stav se prostě nevytvoří a uživatel má poté nesprávně vytvořenou strukturu. Je možné ale strukturu modifikovat v grafickém editoru, takže není nutné aby uživatel začínal od začátku.

Když je tedy stav v jádře vytvořený, projde se řádek tabulky, který koresponduje s pořadím stavu. Program má již představu o počtu stavů, událostí, výstupů a proměnných, takže ví přesně v které části tabulky má hledat. Nejprve se vytvářejí přechody. Jakmile algoritmus narazí na přechod, podívá se ke které události patří a vytvoří ve stavu přechod, jehož identifikátorem ve slovníku bude zkratka události a jako hodnotu bude obsahovat zkratku stavu, do kterého má automat přejít. Hlídá se, zda přechod obsahuje další stav, nebo metodu (například může obsahovat metodu *DeviceReset()*). Tyto dvě možnosti se ukládají ve struktuře zvlášť, protože přechod může obsahovat oboje, nebo jen jednu z nich, záleží na systému stavového automatu. Když je přechod vytvořený, algoritmus prohledá slovník událostí, zda obsahuje událost se zkratkou, která se použila pro vytvoření přechodu. Pokud ne, vytvoří událost ve slovníku s jejím přesným názvem. Pokud ano, krok se přeskočí. Tímto způsobem si aplikace sama vytváří ostatní slovníky a stačí jen dát pokyn k vytvoření slovníku stavů. Stejným způsobem se ve stavu vytváří hodnota výstupu. Ve slovníku výstupů jsou pouze uvedeny výstupy které jsou v logické jedničce, tedy v hodnotě true. Ostatní výstupy v logické nule se doplňují automaticky až je tento případ vyžádán testovací procedurou. Slovník výstupů se vytváří přesně jako slovník událostí. Vyjímkou jsou proměnné, protože ty svůj vlastní zvláštní slovník ve struktuře nemají a to protože jsou pouze jako referenční hodnota pro určitý stav a není potřeba k nim popřípadě přiřazovat metodu, kterou bude nutné volat. Na druhou stranu bude spíše vstupní hodnotou pro některé metody, které se budou ve stavovém automatu volat. Proměnná se do stavu ukládá pod svojí zkratkou a obsahuje dále své celé jméno a hodnotu ve formátu double.

Tímto způsobem se vytvoří celá struktura stavového automatu v jádře. Na konci tohoto procesu lze říci, že v jádře jsou veškeré informace, které potřebujeme pro grafické znázornění stavového automatu. Struktura v jádře se pomocí příkazů *Serialization* převede na XML dokument. Tyto příkazy jsou k dispozici přímo v prostředí C#. Celý dokument je uložen ve složce Temp v adresáři aplikace. Následně jsou všechna data v dočasných slovnících v přístupové vrstvě smazána a také je smazána struktura v jádře aplikace. Tímto je aplikace připravená na případné zpracování další struktury. K tomuto případu ale nedojde, protože se ihned začne s vykreslením stavového diagramu v grafickém editoru.

2.5 Grafický editor

Pro vykreslování stavových diagramů byl použit již vytvořený projekt Diagram.Net, který je volně dostupným nástrojem pro vytváření diagramů. [2]

Hlavní problém celé knihovny pro vykreslování diagramů byla absence jakéhoko-

liv manuálu, nebo aspoň příkladů jak s touto knihovnou pracovat. Další části této kapitoly se z drtivé většiny zaměřují na vyřešení těchto problémů.

Kvůli jednoduchosti a přehlednosti byla základní koncepce stavového diagramu vytvářeného aplikací, která je tématem této semestrální práce zvolena tak, že grafická část bude obsahovat pouze stavy reprezentované obdélníky a přechody mezi nimi.

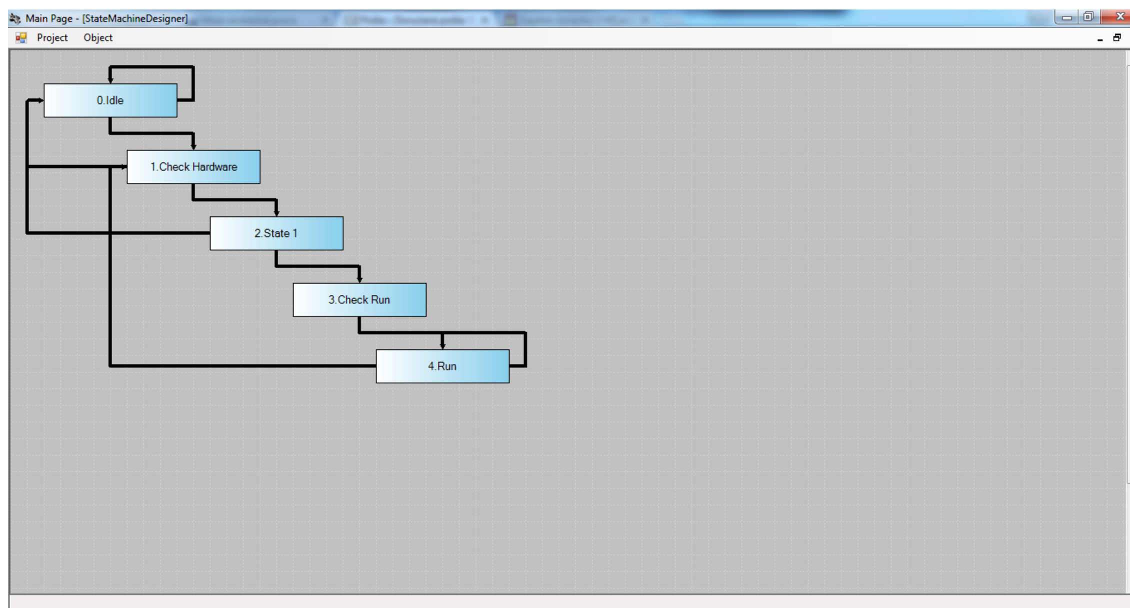
Znovu byl zvolen poloautomatický systém vytváření objektů, který ale bude mít uživatel možnost modifikovat. Některé detaily, které úplně přesně nevykreslují diagram, byly ponechány tak, aby je uživatel sám upravil, pokud to bude výslovně dle jeho představ nutné. Je potřeba ale dodat, že většina problémů, které bylo nutné vyřešit bylo dáno samotnou knihovnou pro vykreslování objektů. Protože ale byla již zvolena, naimplementována a rozpracována, bylo časově výhodnější vyřešit tyto problémy, než hledat náhradu, nebo vytvářet vlastní knihovnu, což by byla pravděpodobně práce na samostatnou diplomovou práci.

Protože je vytvářená aplikace postavená na systému MDI, bylo možné ponechat zobrazovacímu oknu celou plochu obrazovky a umožnit tak co nejvíce komplexní přehled o stavovém diagramu, bez jakýchkoliv rušivých elementů. Na obr.2.6 je zobrazena pracovní plocha pro uživatele, pokud chce vytvářet své vlastní diagramy. Pracovní plocha v tomto případě zobrazuje část řídídího stavového automatu, který je zde pouze pro názornou ukázku, jak může pracovní plocha aplikace vypadat v určité části rozpracovaného projektu. Lze zde vidět, co již bylo řečeno výše. Drtivá většina operací se provádí v hlavním okně z rozbalovacího menu a interakce s objekty, které budou vkládány do pracovní plochy se provádí označením objektu a pravým tlačítkem myši se objeví další možnosti, jak s objektem pracovat. Pracovní plocha grafického editoru je dělená čtvercovou sítí, která usnadňuje orientaci a umožňuje programu lépe indexovat pozice jednotlivých objektů.

2.5.1 Vykreslení jednotlivých objektů

Jak již bylo řečeno, stav bude v grafickém rozhraní reprezentován jako obdélník. V knihovně Diagram.NET je takovýto objekt pojmenován jako *RectangleNode*. V aplikaci je základní obdélník definován určitými rozměry, takže aplikace bude vytvářet objekty s přesně definovaným rozměrem. Samotný objekt *RectangleNode* má osm uzlů, do kterých je možné připojit čáru, která reprezentuje přechod. Objektem, který bude tento přechod vizualizovat je *RightAngleLinkElement*. Pro přehlednost aplikace bude objekt *RectangleNode* využívat pouze čtyři uzly, rohové nebudou využity.

Na obr.2.7 je zobrazeno okno, které slouží k vytvoření stavu, pokud jej chce uživatel sám definovat. Toto okno je dostupné přes menu Object->Add state v hlavním

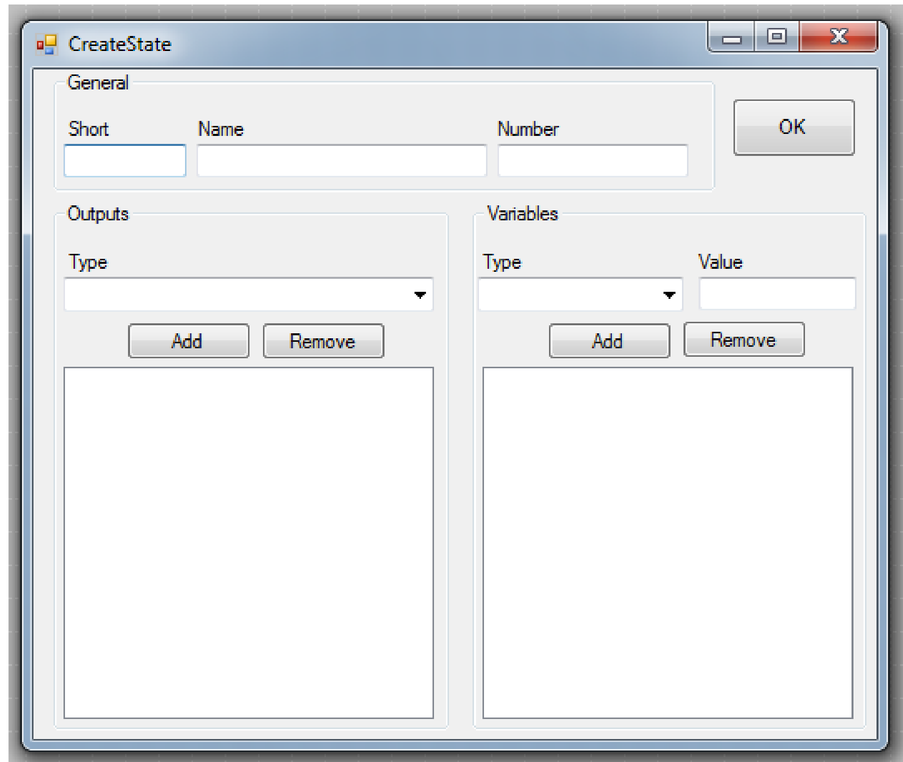


Obr. 2.6: Pracovní plocha uživatele

menu aplikace (MDI hlavní formulář). Okno obsahuje veškeré informace a ovládací prvky, které uživatel potřebuje pro vytvoření stavu do stavového diagramu. V levém horním rohu okna jsou obecné informace o stavu, tak jak jsou definovány ve stavové tabulce a ve spodní části uživatel přidává výstupy a proměnné, tak jak jsou nadefinované. Pokud vytváří prázdný projekt, musí si všechny výstupy a proměnné nadefinovat. Uživatel ale nutně nemusí vyplňovat výstupy a proměnné, může je nechat nevyplněné a doplnit je později. Na obr.2.8 je výřez z pracovní plochy grafického editoru se dvěma stavy. Pro příklad jsou zde dva stavy, jeden s názvem Odin a druhý s názvem Thor.

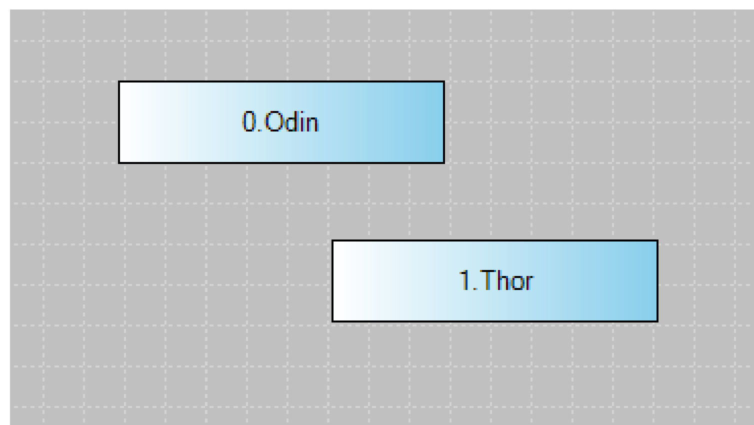
Vytvořený stav je do pracovní plochy vložen na jednu a tutéž pozici v pravém horním rohu, kde se předpokládá, že bude mimo stavový diagram (za předpokladu že většinou umisťujeme objekty do levé části pracovní plochy, ale není to pravidlo, záleží na uživateli). Takto vytvořené stavy lze libovolně přesunovat a měnit jejich velikost. Základní velikost ale postačuje pro většinu jmen, kterými se stavy označují.

Pro přidání přechodu mezi stavy slouží menu Object->Add transition. Uživatel následně musí zvolit stav ze kterého chce přejít a stav do kterého chce přejít. V levém spodním rohu okna je řádek, který uživatele navádí co má následně udělat. Jakmile vybere druhý stav, aplikace ví které dva stavy má propojit a následně zobrazí uživateli okno, kde musí vybrat událost, která přechod vyvolá. Jako v předchozím případě, pokud uživatel nevytváří stavový automat ze stavové tabulky, musí si veškeré přechody nadefinovat sám. V případě vytváření události pouze stačí zadat zkratku a název události, další potřebné informace již aplikace zná.



Obr. 2.7: Okno pro vytvoření stavu

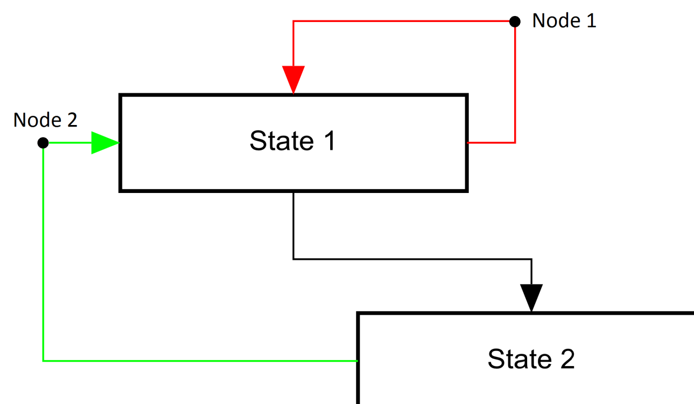
Při vykreslování přechodů se narazilo na problém, který je způsobený samotnou knihovnou Diagram.NET. Princip vytváření elementu *RightAngleLinkElement* je takový, že se počítá nejkratší cesta, která vede z jednoho uzlu do druhého. Tento princip značně znepřehledňuje celý stavový diagram, pokud v něm máme více stavů (v testovaných příkladech se vyskytovalo více než deset stavů). Dalším velkým problémem bylo vyobrazení smyčky, kdy například po vypršení času stavový automat nepřejde do jiného stav, ale vrátí se do původního. Tento způsob se používá například u stavu Idle, kdy se čeká na signál od výstupu a systém přejde do dalšího stavu. Takovýto přechod by v editoru byl zobrazen *za* stavem, který jej vyvolal, protože podle knihovny Diagram.NET se počítá nejkratší cesta, která je pro jakékoliv kombinace čtyř uzlů ve stejném stavu schovaná za samotným tělem stavu. Vytvoření přechodu ve stejném uzlu by přineslo pravděpodobně ještě horší výsledky. Stejný problém nastal i u dalších přechodů, kde nebylo jasné kde přechod přesně vstupuje do stavu (ve kterém uzlu) a to v případě kdy přechod procházel okrajem stavového obdélníku. Vytváření přechodů nejkratší cestou dále způsobilo že mnoho přechodů vedlo v grafickém editoru takovou cestou, že se překrývaly s mnoha objekty a celkový dojem stavového diagramu byl nepřehledný. Pokud by uživatel musel každý takovýto přechod ručně přesunout, zabralo by to u složitého diagramu znatelné množství času.



Obr. 2.8: Vytvořené stavy

Na obr.2.9 je ukázka, jak byl problém vyřešen. Přidáním dvou elementů *ElipseNode* ke každému stavu a napojením přechodů *RightAngleLinkElement* na tyto elementy, které slouží jako pomocné body bylo docíleno zobrazení, které bylo mnohem přehlednější pro uživatele. Nevýhodou tohoto řešení je to, že se přechod nebude skládat z jednoho elementu, ale ze dvou. Dalším problémem bylo to, že element *ElipseNode* sice obsahuje také šest uzlů, takže by teoreticky bylo možné využít uzly k oddělení jednotlivých přechodů (alespoň od sebe oddělit přechody určitého typu), ale v průběhu testování se ukázalo, že takovýto způsob není možný a vykreslování přechodů bylo nepřesné (například odskoky o zlomek do strany, kdy čára měla být rovná) a nebylo se jich možné zbavit ručním přesunem. Z tohoto důvodu nakonec vyplynulo, že v jednom případě, pokud do stavu vstupuje a vystupuje mnoho přechodů určitého typu, může se stát, že při označení přechodu dojde k označení většího počtu dalších stavů. které uživatel neměl v úmyslu označit. Protože ale knihovna neumožňuje popsat každý přechod, aby se dal identifikovat (respektive umožňuje, ale nejdou odstranit popisy přechodů, ani po pokusu toto obejít nebo to vymazat ze samotného zdrojového kódu) nelze se tohoto problému jednoduchým způsobem zbavit. Tato událost ale byla jistým způsobem ošetřena dále. Na obr.2.9 jsou pomocné body popsány jako Node 1 a Node 2. Červeně je vyznačen přechod, který znamená, že automat se neustále vrací do stejného stavu.

Identifikace jaký pomocný bod se má použít se děje na základě vzdálenosti jednotlivých stavových obdélníků. Lze znovu použít obr.2.9. Každý objekt *RectangleNode* má svoji základní pozici, ze které se vykresluje. V tomto případě je to levý horní roh objektu. Mohou nastat čtyři přechody - prvním je smyčka vracející se do stejného stavu. Pokud se pozice vybraných objektů shodují, znamená to že se má vytvořit takováto smyčka, tedy že se mají vytvořit dva přechody přes pomocný bod Node 1. Dalším typem přechodu je přechod mezi nejbližšími stavy. V tomto případě



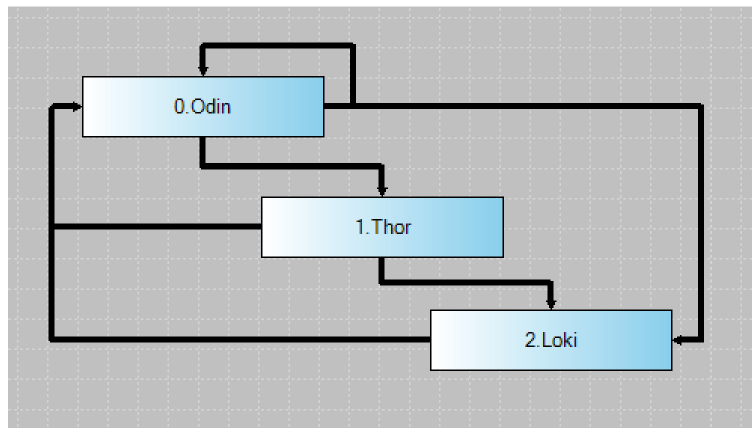
Obr. 2.9: Vykreslování přechodů

lze předpokládat, že stavy leží blízko u sebe a je zde zvolená hraniční hodnota, od kterého již aplikace nebude stavy brát jako sousední. Pokud nastane případ, že stavy leží v této toleranci, vytvoří se pouze jeden přechod, jak je naznačeno na obr.2.9 mezi stavy State 1 a State 2.

Třetím typem přechodu, je pokud následující stav leží dále než je tolerance blízkého stavu. V tomto případě se vytvoří také dva objekty *RightAngleLinkElement* a pomocným bodem bude také Node 1. Posledním případem je, pokud se přechází zpětně - tedy do stavu který je výše ve stavovém diagramu. V tomto případě se vytvoří také dva elementy *RightAngleLinkElement*, ale pomocný bod bude Node 2. Dále je ještě zajištěno, že přechody které vedou z pomocných bodů do stavu ke kterému náleží, se vytvářejí pouze jednou, aby se zbytečně nevytvářely vícekrát.

Příklad reálného vykreslení je na obr.2.10. Byl použit příklad ze začátku této kapitoly, pouze přibyl třetí stav Loki. Lze zde vidět veškeré použité přechody a také to, že jsou vytvořené dle předpokladů, které byly určeny na obr.2.9. Pokud tedy uživatel bude potřebovat ručně vytvářet stavy a přechody mezi nimi, je mu umožněno provést to v kompletní míře. Přechody se vytvářejí automaticky a není nutné je ručně vést a určovat ze kterého uzlu mají vycházet. Tento způsob by mohl být považován za nedostačující, ale pokud uživatel objeví způsob vytváření přechodů, může s nimi již dopředu počítat a vyhnout se tak zbytečnému odstraňování a novému vytváření objektů.

Další výhodou tohoto zpracování přechodů je, že takovým to způsobem je možné automaticky generovat stavový diagram a vyhnout se tak například počítání nejkratší možné cesty, která je v některých případech nevýhodná, jak bylo řečeno výše. Tímto tématem se zabývá následující kapitola.



Obr. 2.10: Reálné vykreslení přechodů

2.5.2 Postup vykreslení při převodu ze stavové tabulky

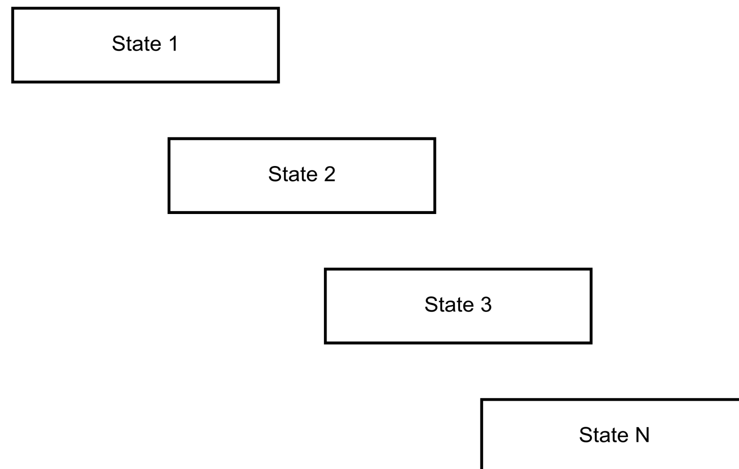
Systém vykreslování přechodů již byl popsán v předchozí kapitole. Tato je zaměřena na systém vykreslování, pokud máme vytvořenou strukturu v jádře ze stavové tabulky. V konečném důsledku je to tedy postup automatizovaného vykreslování stavového diagramu.

Jakmile je vytvořená struktura v jádře a uložena ve formátu XML dokumentu ve složce Temp a provede se vyčištění všech struktur, nahraje se dočasný soubor zpět do jádra a spustí se automatické vykreslování.

Prvním krokem si aplikace zjistí kolik stavů vůbec stavový automat obsahuje a pak jednotlivé stavy začne vykreslovat spolu s informacemi, které jsou v určitém stavu obsaženy, tedy s aktivními výstupy a proměnnými. Stavy se vytvářejí do schodovité struktury, která je naznačena na obr.2.11. Tato struktura byla zvolena na základě toho, že stavů je v drtivé většině stavových automatů reálných zařízení mnoho a aby se plocha diagramu rozložila do celé pracovní plochy grafického editoru.

Jakmile jsou stavy vykreslené, přejde se na vykreslení přechodů. Znovu se postupuje stav od stavu a procházejí se slovníky přechodů přímo v každém stavu a podle vzdáleností jednotlivých objektů se volí cesty (postup byl popsán v předchozí kapitole). Z tohoto důvodu byly nejdříve vykresleny stavy, protože přechod potřebuje protější stav a bylo by obtížnější vykreslovat stav úplně na konci, když je pouze vytvořený první stav, nehledě na to že by bylo nutné kontrolovat zda již stav není vytvořený a podobně.

Příklad vytvořeného stavového diagramu bude uveden v kapitole **??**. Protože se v uvedené kapitole bude popisovat převod celého jednoho projektu (kapitola by měla sloužit jako jakýsi manuál), není nutné zde popisovat a zobrazovat jednu a tutéž věc dvakrát.



Obr. 2.11: Rozložení stavů při automatickém vykreslování

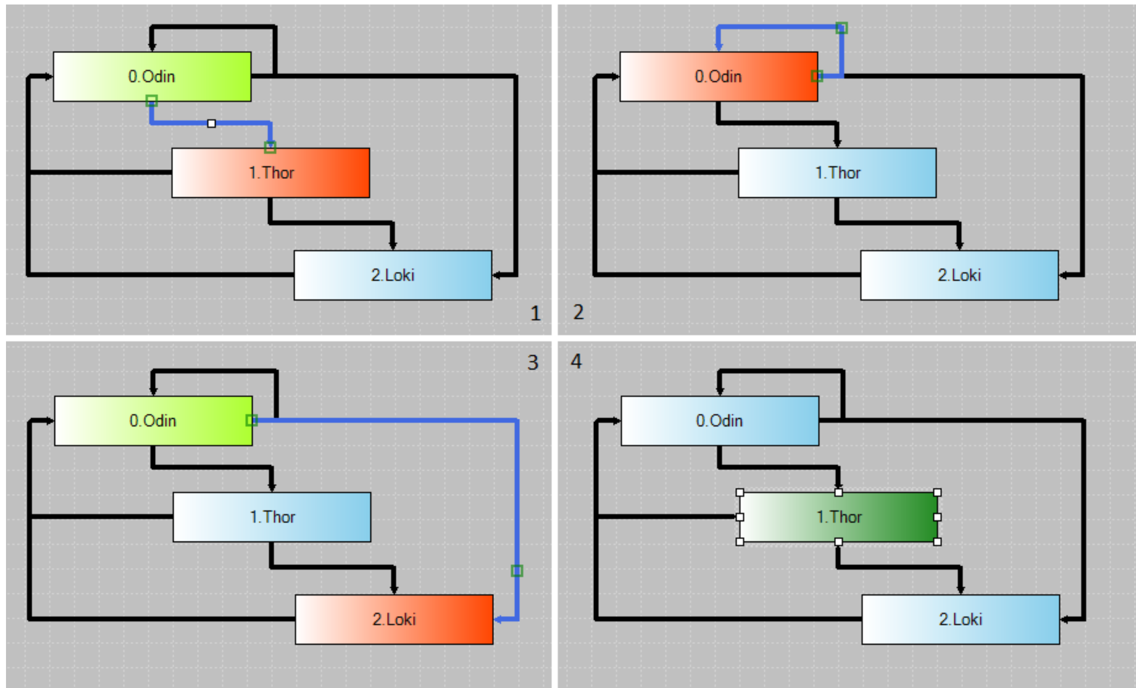
2.5.3 Ostatní ovládací prvky editoru

Mezi další ovládací prvky, které jsou dostupné v editoru patří editace, zobrazení informací, odstraňování a nakonec označování. Poslední položka je pouze pro vizuální přehlednost.

Při editaci stavu, je nutné nejprve označit levým tlačítkem myši stav, který chce uživatel editovat a následně pravým tlačítkem myši vyvolat menu a zvolit položku Edit. Zobrazí se stejné okno, jako je v případě vytváření stavu na obr.2.7, pouze s rozdílem, že není možné změnit položky jméno, číslo a zkratka stavu, ale ostatní položky měnit lze a tedy je možné přidávat a odstraňovat výstupy a proměnné. Editace není možná u přechodů, tam se prakticky provádí přímo odstranění přechodu a jeho vytvoření na jiném místě. Tento způsob pro přechody byl zvolen pro eliminaci přemíry ovládacích prvků, které by v konečném důsledku nebyly příliš efektivní, protože by bylo nutné překreslovat celý přechod prakticky stejným způsobem, který probíhá při mazání a novém vytváření.

Stejně tak u přechodu není možnost Info. Tento příkaz lze znovu vyvolat přes pravé tlačítko myši u označeného stavu a to z toho důvodu, že informace o přechodech jsou obsaženy ve stavech a možnost Info zobrazuje i přechody, které vedou z označeného stavu a události, které přechod vyvolávají.

Pokud se tedy zvolí přechod a použije se pravé tlačítko myši, uživateli je umožněna pouze možnost mazání stavu. Při této operaci se přechod odstraňuje ze struktury v jádru aplikace a to ze stavu, ve kterém je obsažen. Stav je umožněno také vymazat a při této operaci je nutné projít celou strukturu stavového automatu a vymazaný stav odstranit jak ze slovníku stavů, tak i ze všech přechodů, které do tohoto stavu vedou.



Obr. 2.12: Příklady označování

Poslední operací je označování, které je zobrazeno na obr.2.12. Jsou zde čtyři možná zobrazení. První, druhé a třetí se týká přechodů a čtvrté se týká označení jediného stavu. Pokud uživatel označí přechod, je mu následně spolu s přechodem zvýrazněno, kterých stavů se týká. Stav, ze kterého se vychází je světle zelenou barvou a konečný stav je barvou červenou. Na zobrazení číslo dvě lze vidět, co se stane, pokud uživatel označí přechod, který vede do stejného stavu (smyčku končící ve stejném stavu). V této části byl objeven problém, který je popsán výše v kapitole 2.5.1, tedy to, že pokud do stavu vstupuje příliš mnoho přechodů, je možné při označení přechodu velice blízko ke stavu (tedy čáry, která vstupuje do pomocného uzlu mimo stav), že jsou zvýrazněny všechny stavy, které do tohoto uzlu vstupují. Je to také dáno problémem toho, že velice blízko u stavu není možné rozlišit jednotlivé přechody. Tímto způsobem by pak bylo možné odstranit více přechodů, než by uživatel zamýšlel a proto zde byla vložena pdomínka, že uživatel nemůže mazat více jak jeden objekt najednou, což platí jak u stavů, tak u přechodů a tedy musí označit přechod v místě, kde se blíží spíše konečnému stavu.

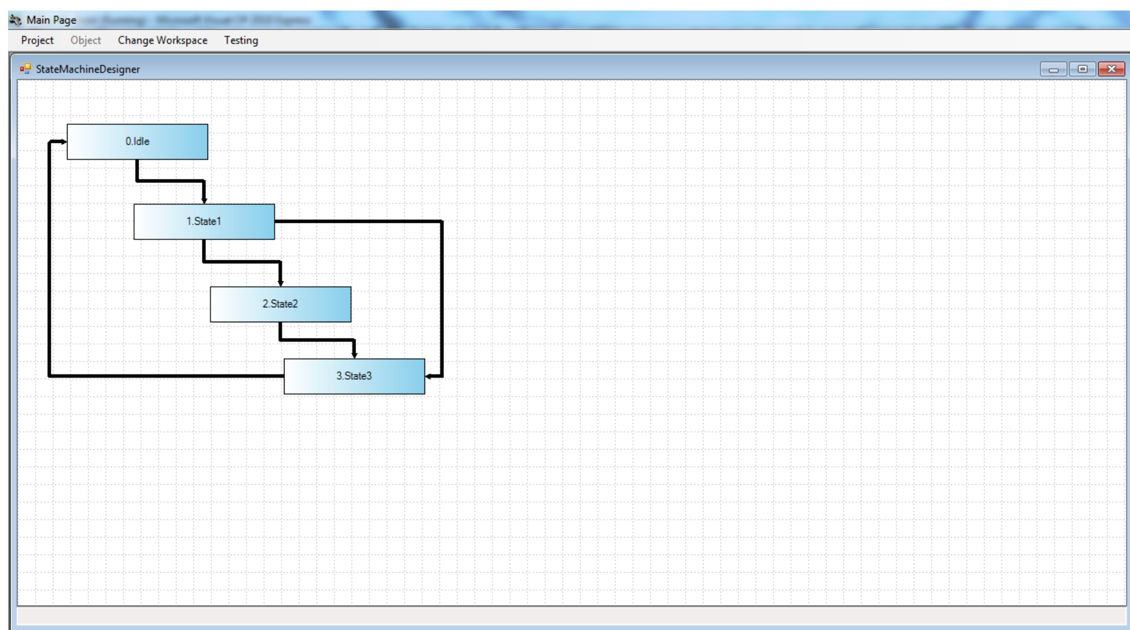
2.6 Testovací prostředí

Pro grafickou prezentaci testovacího prostředí je použit grafický editor vytvořený pro editaci a úpravu stavových diagramů. Samotné testování ale vyžaduje pouze

grafickou prezentaci průběhu testů. Další editační zásahy by znamenaly úpravu za běhu programu a zapříčinily by havárii celé aplikace, jelikož by se některá data mohla změnit nežádoucím způsobem pro testovací proceduru.

Proto byl grafický editor modifikován, aby nebylo možné jakkoliv změnit jeho podobu, která by ovlivnila testování. Pro tuto změnu slouží ovládací prvek *Change Workspace*, kde je možné přepínat mezi editorem a testovacím prostředí. Vizuální kontrola, ve kterém prostředí se uživatel nachází, je provedena jak modifikací ovládacích prvků, tak změnou pozadí okna ze šedé na bílou. Uživateli jsou zpřístupněny možnosti pro testování a úpravu funkcí, které slouží pro komunikaci se zařízením a na druhou stranu jsou mu znemožněny úpravy grafického editoru zmíněné v předchozí kapitole, včetně akcí které lze ovládat stisknutím pravého tlačítka myši.

Na obr.2.13 je výřez z okna aplikace, která je přepnutá do testovacího prostředí. Jsou zde vidět hlavní grafické změny popsané v předešlém odstavci. Celé nastavení prostředí, jak grafického tak testovacího se ukládá do jádra aplikace - do stromové struktury, takže po uložení nedojde ke ztrátě dat.

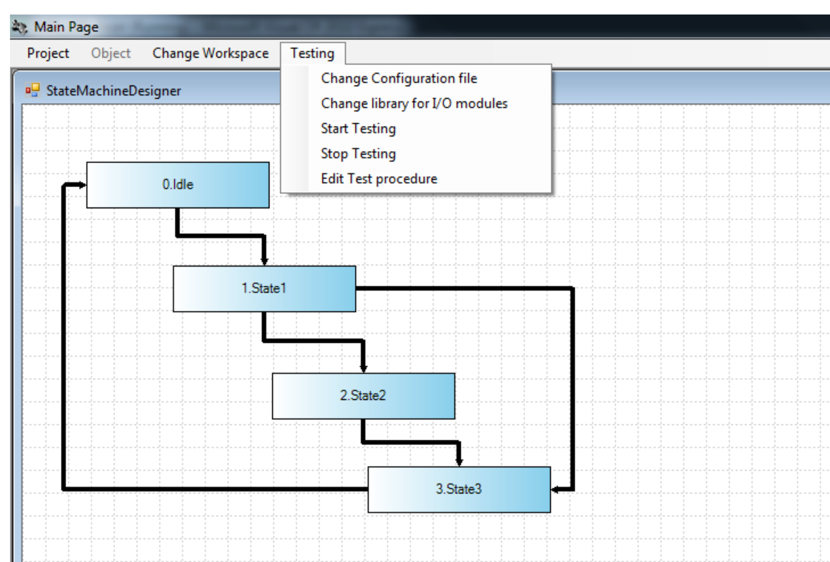


Obr. 2.13: Testovací prostředí

Takto upravené grafické prostředí dále informuje uživatele o průběhu testu, neboli v které části stavového diagramu se zařízení nachází. Protože ale není známá přesná vnitřní informace o stavu zařízení a aplikace je odkázaná pouze na posloupnost akcí a reakcí na příslušné vstupy, není nikdy jisté, zda se řídicí jednotka testovaného zařízení v tomto stavu opravdu nachází. Tento typ testování se zaměřuje pouze na takzvaný 'Black box' test, kdy nejsou známé přesné vnitřní procesy řidící jednotky, ale je popsané její chování na základě vnějších stimulů. Na základě posloupnosti akcí

lze ale prakticky ve všech případech dohledat a určit, zda stavový diagram odpovídá řídící jednotce. Vždy zde ale může nastat chyba uživatele, který vytvořil aplikaci na základě specifikace zařízení a udělal v některé části návrhu chybu, nebo řídící jednotka neodpovídá přesně specifikaci. Toto rozhodování je již na uživateli, který by měl mít k dispozici podrobnější výsledek testu.

Obr.2.14 obsahuje výřez předchozího okna z obr.2.13 se zobrazenými možnostmi pro uživatele. Lze zde vidět, že je možné změnit umístění konfiguračního souboru a knihoven pro I/O moduly o kterých bude řeč dále v podrobnějším rozebrání testovacího prostředí. Dále umožňuje spustit test, nebo jej přerušit. V druhém případě je test ukončen a uživatel musí počítat s textovým souborem o výsledku testu, který nebude obsahovat kompletní informace. Poslední možností je úprava ostatních nastavení testu, jako jsou použité funkce, definice vstupních portů a další nastavení.



Obr. 2.14: Ovládací menu testovacího prostředí

Další prvky prostředí jsou již prováděny aplikací na základě návrhu uživatele. Samotný návrh diagramu je velice komplexní, stejně jako nastavení testovací procedury, kdy je nutné vložit velké množství informací na jejichž základě je proveden test. Je nutné si uvědomit, že není možné podchytit zda je návrh proveden správně již od začátku a drtivá většina chyb se projeví až při samotném fyzickém testování. Stavové diagramy složitějších aplikací je proto doporučené vytvářet v menších krocích, kdy je vytvořeno více projektů, z nichž každý otestuje určitou část stavového automatu. Samotná aplikace je schopná ovládat a pracovat s rozsáhlými diagramy, ale pokud jsou složitější, je velice jednoduché udělat chybu.

Aplikace a princip testování dále neumožňuje provádět cyklování testů a vytvářet podmínky na jejichž základě se poté postupuje dále. Například nelze pomocí

grafického editoru přidat jakýsi čítač, který by počítal, kolikrát zařízení projde určitým stavem a po dosažení limitní hodnoty čítače přejde do jiného stavu. Pro tento případ lze ale vytvořit stavový diagram, kde budou jednotlivé fáze navrženy hned za sebou, ne ve smyčce a poté bude možné test spustit. Tento fakt je způsoben tím, že aplikace pouze reaguje na odezvy zařízení a srovnává je s tím co uživatel navrhl. Takže pokud se správným způsobem vše nadefinuje bez vzniku konfliktů, je možné testovat i takto, ale je nutné pro určité zařízení navrhnout specifický stavový diagram a z toho vyplývající stavový automat.

V dalších podkapitolách bude rozebrán samotný vývoj této části aplikace, zvolené postupy, možnosti konfigurace, popis funkce a struktura prostředí. Z popisu by mělo vyplynout, proč se testovací prostředí chová způsobem výše popsaným.

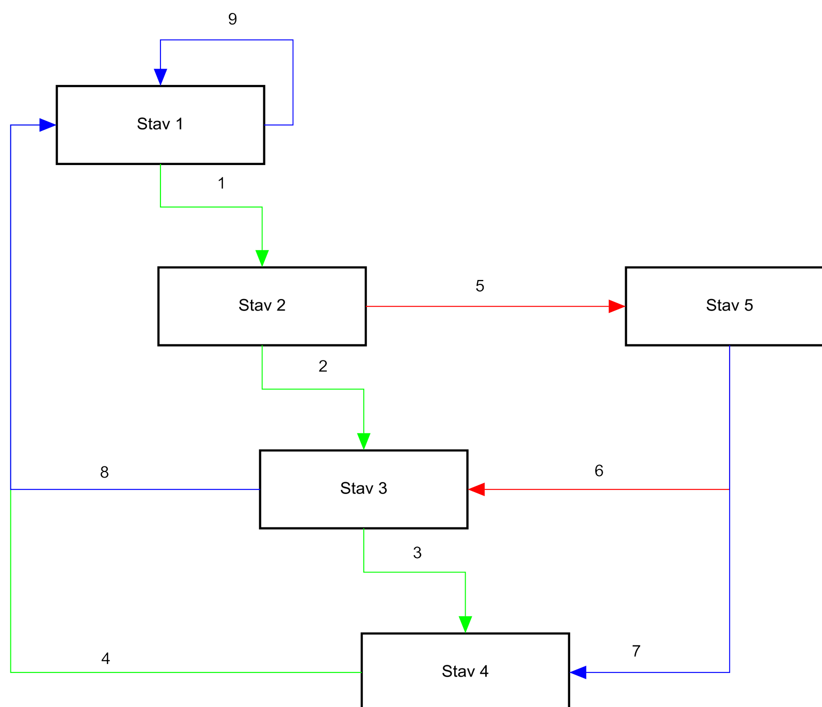
2.6.1 Základní princip testování

Testování stavových automatů je relativně komplexní odvětví, které je množné pojmout z více úhlů. Pokud se zaměříme na nejzákladnější princip, jedná se o princip akce a reakce. Tedy že na určitou akci očekáváme přesně definovanou reakci a jakmile tato podmínka není splněna, je možné prohlásit že test neproběhl dle předpokladů. Na tomto základním principu je postaven celý testovací algoritmus této práce.

Akcí, kterou aplikace vyvolá, je míněna událost která vede ke změně. Může to být aktivní logická hodnota digitálního výstupu, vypršení časového intervalu a podobně. Testovací algoritmus může pracovat s jakoukoliv funkcí kterou mu uživatel nadefinuje, nemusí to být pouze uvedené dva příklady. V první řadě je tedy vyvolána reakce a algoritmus očekává odezvu. Protože v řidících automatech zařízení je kladen důraz na časování, je nutné kontrolovat časový úsek, který je definovaný specifikací zařízení a po vypršení tohoto úseku konstatovat, že reakce nebyla korektní. Možnosti které je nutné podchytit jsou tedy chybné reakce a vypršení časového úseku.

Jak již bylo zmíněno, jedna část testovacího algoritmu se stará pouze o vyvolání akce a kontrolu reakce. Dále je nutné specifikovat, co přesně od testovacího algoritmu očekáváme při obecnějším náhledu. Na obr.2.15 je příklad obecného stavového diagramu, který má být testován. Kritériem pro provedený test může být jak úspěšné provedení všech událostí, tak že se test neprovádí přesně podle specifikace zařízení, nebo také zda existují další události, které nejsou popsané ve specifikaci a jsou nežádoucí. Také lze testovat chybové události, příkladem může být trvale aktivní vstup do zařízení (pro aplikaci je toto trvale aktivní výstup), kdy je žádoucí aby se tento vstup nejdříve deaktivoval a následně aktivoval, ale i přes tento fakt zařízení přejde do dalšího stavu. Testovací algoritmus v této práci se zabývá otestováním všech přechodů, tedy potvrzením, že navržený stavový diagram popisuje řídící automat zařízení. Návrh diagramu nicméně umožňuje do určité míry testovat i netečnost za-

řízení na nežádoucí přechody. Může to být ověřeno například přidáním události, že při aktivaci určitého vstupu zařízení řidící automat zůstane ve stejném stavu.



Obr. 2.15: Stavový diagram a přechody mezi stavy

Testovací algoritmus musí tedy být schopen projít veškeré události stavového diagramu. Je nutné si uvědomit, že testované zařízení nemůže libovolně přecházet mezi všemi stavy zároveň. Procházení diagramu probíhá v jakýchsi testovacích cyklech. Na obr.2.15 je barevně označeno několik takovýchto cyklů. Zeleně je označena takzvaná *hlavní cesta*, která označuje základní smyčku zařízení. Ta může popisovat hlavní princip funkce zařízení (zapálení plamene a začátek ohřevu, spuštění montážní linky nebo start, průběh letu a přistání dopravního letadla). Červeně označené přechody označují takzvanou *chybovou cestu*, která odbočuje z *hlavní cesty* z důvodu nesprávné funkce periferií, které zařízení ovládá (plamen nebyl zapálen, motor linky se nespustil, podvozek nelze vysunout) a modrou barvou jsou označeny další možné odbočky ve stavovém diagramu.

Kdyby bylo možné přejít ze stavu do kteréhokoli jiného stavu, bylo by možné v algoritmu hlídat pouze zda všechny události byly provedené a reakce na ně jsou správné. Protože to ale není u stavových automatů v drtivém případě možné, je nutné aby aplikace byla schopná procházet diagram a čekat dokud nenarazí na událost, která nebyla provedena. Tento způsob rozhodování je klíčový, protože stavové automaty obsahují oddělené větve, které se mohou vracet zpět do hlavní cesty, nebo

pokračovat do chybových cest. To co bylo výše popsáno, lze nazvat jakýmsi sériovým postupem, kdy zařízení neumožňuje paralelní zpracování a k určité akci je nutné provést kroky, které již mohly být provedeny. Nejvyšší prioritu v rozhodování testovacího algoritmu mají události, které ještě nenastaly. Jakmile na takovouto událost algoritmus narazí, automaticky ji provede. Takováto akce jej zavede do dalšího stavu, kde se znovu nejprve hledají události které ještě neproběhly. V tomto prvním cyklu se testovací algoritmus de facto 'naučí' základ stavového diagramu. Na obr.2.15 můžeme jako příklad uvést, že hned po spuštění algoritmus postupně prošel zelenou *hlavní cestu*. Vrátil se do stavu číslo jedna a bylo zjištěno, že je zde další událost (číslo devět), která ještě nebyla provedena. Tímto se ale dostane znovu do stavu číslo jedna. Všechny události, které se vztahují k tomuto stavu byly již provedeny, ale algoritmus si udržuje počet všech stavů a tedy 'ví' že neprovedl vše.

V této fází testu bylo nutné rozhodnout, jakým způsobem postupovat dále. Jednou možností bylo, že bude zkoušet všechny události znovu. V tomto případě by bylo nutné řešit kolikrát se která událost znovu provedla a udržovat tuto informaci v patrnosti a dále vybírat události, jejichž počet použití je nižší. Zde může nastat problém, že do některých stavů zařízení přejde mnohem častěji než do jiných. Tímto stavem může být například výchozí stav číslo jedna. Jak již bylo řečeno výše, postup diagramem na základě chování řídících jednotek zařízení je sériový a tím pádem zařízení provádí cykly. Tyto cykly budou v drtivé většině začínat ve výchozím stavu. Popsaný způsob má výhodu v tom že dříve nebo později opravdu projde všechny události navržené ve stavovém diagramu, délka testování se ale tímto způsobem prodlouží a velké množství událostí bude prováděno znovu, což může být v určité fázi považováno za neefektivní. Z tohoto důvodu byl vybrán odlišný způsob procházení již 'naučeného' stavového diagramu.

Principem je, že pokud jsou všechny události v aktuálním stavu provedené, testovací algoritmus vyhledá událost která vede do nejbližšího stavu (je tím míněno do číselně nejbližšího stavu). Vycházelo se z faktu, že základní *hlavní cesta* zařízení je vytvářena uživatelem jako první a tudíž mezi stavy jsou nejkratší vzdálenosti. V tomto případě je termín *vzdálenosti* pouze informativní aby bylo možné princip popsat. Tímto postupem je zajištěno, že algoritmus bude procházet *hlavní cestu* a hledat odbočky z tohoto cyklu. Tímto způsobem lze rychle odhalit události které vedou do dalších větví a takzvané *chybové cesty*.

Popsaným způsobem ale nelze otestovat události, které jsou mimo hlavní cyklus a mimo hlavní odbočky. Příkladem může být na obr.2.15 přechod číslo sedm. V průběhu testu aplikace provedla zelený hlavní cyklus, vedlejší červený chybový cyklus a většinu modrých událostí. Jakmile ale je provedena většina událostí a zbývá poslední, která je náhodou ve vedlejší větvi diagramu, algoritmus by cykloval nekonečně v hlavním cyklu a nikdy by neskončil, protože podmínkou k úspěšnému

dokončení testu je nutné provést všechny události. Proto bylo nutné hlídat, zda se algoritmus nezacyklil ve smyčce, ze které nemá šanci bez vnějšího zásahu vystoupit. Z tohoto důvodu je nutné hlídat, kolikrát byla smyčka provedena. Jelikož se již v tomto případě algoritmus zaměřuje na definovaný problém, je možné nastavit hranici rozhodování relativně nízko a ze smyčky vyskočit jiným způsobem. V tomto případě byla vyzkoušena možnost, kdy algoritmus nehledá nejkratší cestu ve stavovém diagramu, ale nejdelší. Tím je možné ze smyčky vyskočit. Můžou ale nastat problémy, kdy nejsou brány v potaz další odbočky mezi nejkratší a nejdelší cestou. při samotném testování algoritmu se tento případ ale neprojevil, ale je možné že v budoucnu tento případ nastane a bude nutné rozhodovací část algoritmu dále rozšířit. V současné situaci je vytvořený algoritmus schopný relativně rychle projít všechny události stavového diagramu.

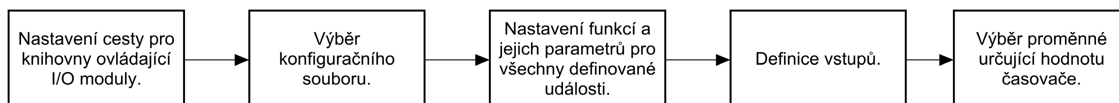
Lze tedy prohlásit, že navržený a vytvořený algoritmus se snaží co nejrychlejším a nejefektivnějším způsobem otestovat veškeré navržené události ve stavovém diagramu a tímto způsobem otestovat řidící automat testovaného zařízení. Algoritmus je možné dále upravovat dle problémů, které mohou dále nastat, záleží na typu a definici problému.

2.6.2 Konfigurace testovací procedury

Jak již bylo řečeno výše, po přepnutí aplikace do testovacího algoritmu je nutné nejdříve zkontrolovat, zda jsou v projektu nadefinovaná veškerá potřebná nastavení pro provedení testu. Nastavení probíhá v jednotlivých krocích, které jsou znázorněné na obr.2.16. Samozřejmě záleží na tom, do jaké míry je projekt nakonfigurovaný. Pokud například budou již známé cesty pro první dva kroky, přejde se přímo ke třetímu kroku atd. Aplikace tedy nejprve potřebuje vědět s jakými nástroji jí bude umožněno pracovat. Jakmile bude znát cestu ke knihovnám, potřebuje mít dále identifikovány I/O moduly. Takovýto konfigurační soubor vlastně popisuje I/O kartu aby se k ní dalo z aplikace přistupovat. Z důvodu, že může být připojeno více I/O karet, s různými vlastnostmi a názvy, je nutné v konfiguračním souboru pojmenovat I/O kartu a přiřadit jí adresu, aby k zařízení mohl program přistupovat. Každá karta obsahuje množství digitálních nebo analogových vstupů a výstupů. Na základě připojeného zařízení musí být v konfiguračním souboru nastaveny jednotlivé piny jako vstupní nebo výstupní a také pro přehlednost je třeba každý fyzický pin pojmenovat specifickým názvem. Tímto souborem je předložen aplikaci kompletní popis I/O karty a pomocí těchto názvů lze postupovat dále.

Jakmile jsou tyto názvy známé a je známá i cesta ke knihovnám, je možné pomocí těchto souborů inicializovat testovací proceduru. Obecně se jedná o vytvoření potřebné třídy v aplikaci, která bude obsahovat potřebné testovací funkce a bude

s jejich pomocí možné fyzicky přistupovat k testovanému zařízení. Procedura ale v této fázi disponuje pouze nástroji, které může použít a ví jak k zařízení přistupovat. Testování je tedy spuštěno až poté co jsou veškeré potřebné konfigurace provedené.



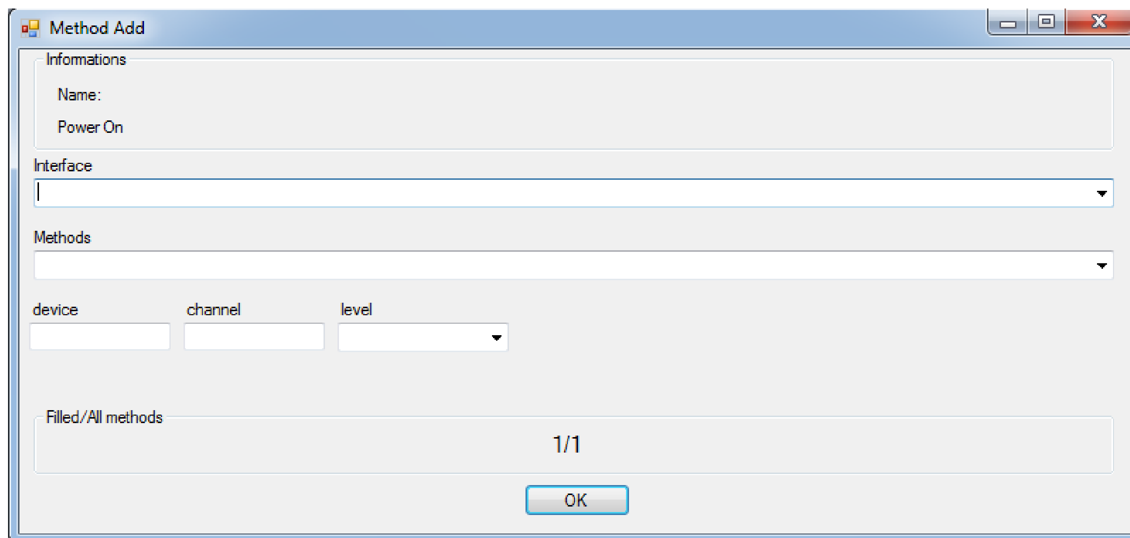
Obr. 2.16: Postup konfigurace

Proto je nutné provést třetí krok z obr.2.16 a tím je přiřazení metody(funkce) každé události, která je definována ve stavovém diagramu. Protože se jedná o funkci, musí být veškeré její parametry uloženy do jádra aplikace, do výše zmíněné stromové struktury a to přesně do slovníku událostí. Aby bylo možné takovouto metodu(funkci) volat, musí mít aplikace k dispozici přesné informace. Použitý programovací jazyk umožňuje použití funkce *Invoke()*, pomocí které lze zavolat libovolnou funkci, která je v určité části nadefinována. K tomu aby ji bylo možné zavolat je nutné znát název funkce, typy vstupních parametrů a jejich hodnoty. Typy parametrů jsou nutné i z toho důvodu, že funkce se mohou nazývat stejně, ale v závislosti na vstupních parametrech se jejich výsledné použití mírně mění. Mnohé funkce jsou totiž vytvořeny tak, aby změnou vstupních parametrů byla mírně změněna i funkčnost. Poté je možné modifikovat pouze jednu funkci a nevytvářet další, jejíž funkce je stejná nebo velice podobná.

Na obr.2.17 je okno, které umožňuje veškeré výše uvedené informace uložit do stromové struktury v jádře. Uživatelské prostředí tohoto okna se vytváří postupně, v závislosti na tom, jaké funkce uživatel vybírá. Na tomto obrázku je vidět již plně vytvořené okno. Je zde vidět že uživatel nejprve musí vybrat prostředí, ve kterém se funkce nachází. Toto se odvíjí od struktury knihovny I/O modulů, kde mohou být funkce děleny dle toho zda se vztahují ke vstupu, výstupu, časování a podobně. Další položkou je výběr samotné metody, která bude volaná aby vyvolala událost. Na obrázku je použitá událost, která připojí napájení k testovanému zařízení. Použitá metoda se bude následně logicky vztahovat k nastavení výstupu na I/O kartě. V závislosti na vstupních parametrech metody se dále zobrazí nástroje, které umožní zadat tyto informace. V zásadě se jedná o identifikaci pinu na I/O kartě, kde je nutné definovat jméno karty a název pinu. Tyto popisy definoval uživatel konfiguračním souborem a na jeho základě se tyto parametry vyplní. Bylo by možné položku která identifikuje kartu nastavit pouze jednou na začátku, ale nikdy není jisté, zda uživatel nepoužije například dvě karty, například když bude chtít testovat

více zařízení najednou. Poslední parametr se týká nastavení logické úrovně výstupu. Protože zde jsou možné pouze dvě možnosti, bylo využito rozbalovacího menu, aby byla eliminována možnost překlepu.

Poslední informace obsažená v tomto konfiguračním okně je počet metod, které již uživatel vyplnil a počet všech metod, která jsou k dispozici. Tato informace se nejvíce projeví na úplném začátku konfigurace, kdy je uživatel informován o tom kolik je celkově všech událostí a které události již byly nakonfigurovány.

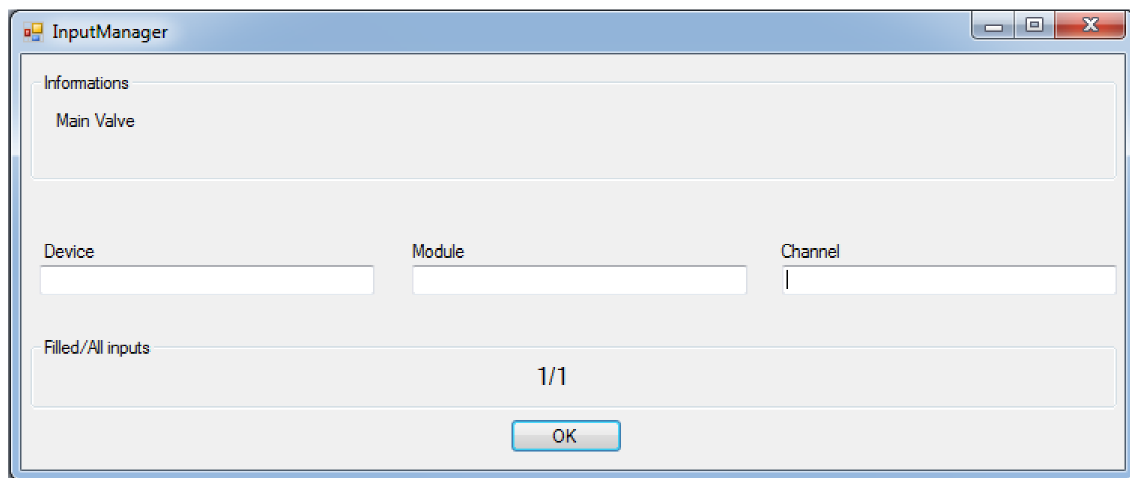


Obr. 2.17: Vložení metody

Největším problémem, který nastal při ukládání těchto informací do stromové struktury bylo to, že veškeré výstupy z těchto ovládacích prvků jsou ve formě textu. Pokud je ale nutné zavolat funkci se vstupními parametry, musí být tyto parametry ve správném formátu. O tento postup se stará nižší vrstva, která převádí vstupní parametry na příslušné typy a ukládá je do stromové struktury v jádře aplikace. Tyto parametry jsou ukládané přímo ve svém formátu a pokud je třeba zavolat určitou funkci aby byla vyvolána událost, přístupová vrstva jádra a hlavní formulář vytvoří dva řetězce, z nichž jeden nese informaci o hodnotách vstupních proměnných a druhý řetězec nese informaci o jejich typech. Tento postup je nutný z důvodu vyhledání potřebné funkce, kdy se toto hledání provádí na základě jejího jména a jako druhý znak slouží již zmíněné typy vstupních proměnných. Poté je již možné funkci zavolat se správnými hodnotami. Konfigurační okno a vrstvy spojující ostatní části aplikace musí zajistit, že se budou vstupní proměnné ukládat ve správném pořadí, aby nedošlo k záměně pozic a tudíž k nezavolání funkce.

Tímto případem byly ke všem existujícím událostem přiřazeny funkce a jejich vstupní parametry. Lze zjednodušeně říci, že se definovaly výstupní porty zařízení a definovaly se akce, které tyto výstupy budou obsluhovat. Obdobným způsobem je

nutné nastavit i vstupní porty aplikace, které se budou následně kontrolovat. Tento případ je naštěstí jednodušší než předchozí krok. Na obr.2.18 je zobrazeno okno aplikace, které tuto konfiguraci provede. Aplikaci je nutné přiřadit k určitému vstupu fyzické adresy z I/O karty. Uživatel musí znovu přesně definovat název zařízení, navíc ještě použitý modul(tedy jméno I/O karty) a samozřejmě pin použitého vstupního portu. Díky tomuto nastavení si aplikace může přesně poskládat použitý vstupní port a zjišťovat jeho hodnotu.

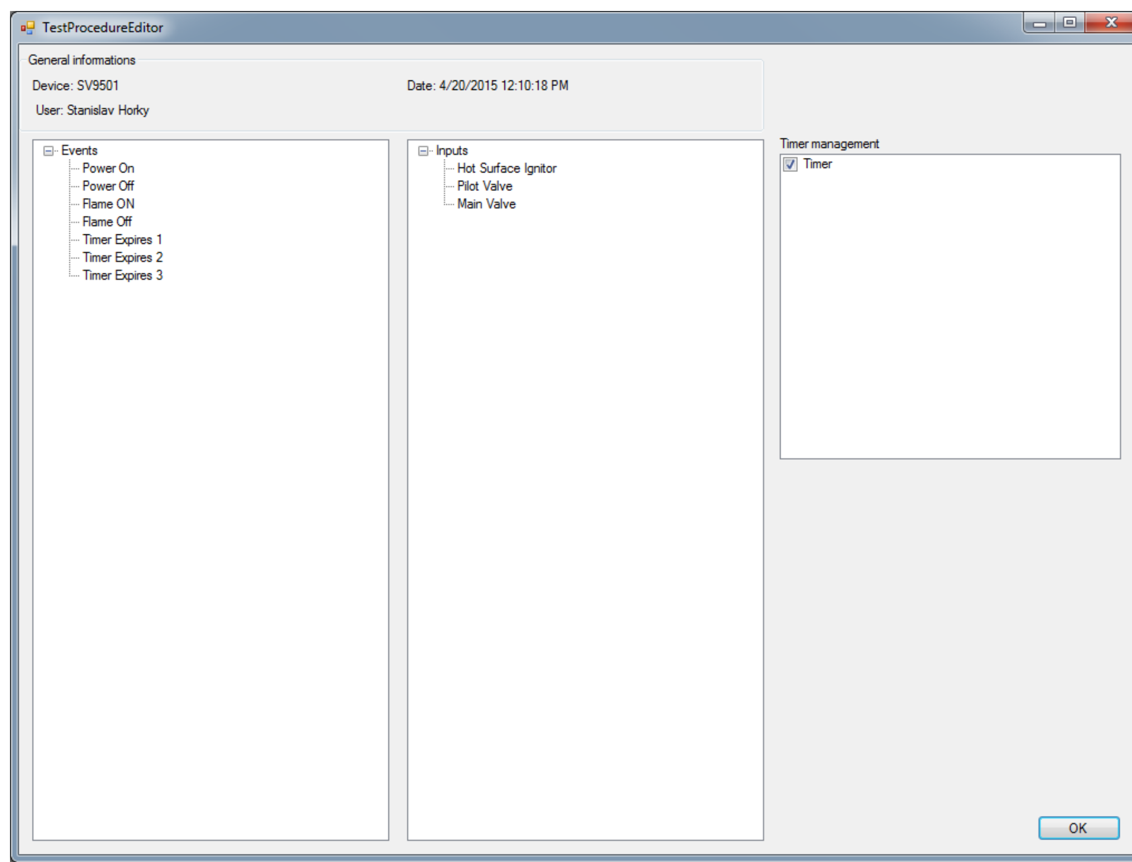


Obr. 2.18: Nastavení vstupu

Nejníže zobrazovanou informací je znovu počet vyplněných a všech definovaných vstupů, které se opět využijí při prvotní konfiguraci testovací procedury a slouží informativně pro uživatele aby bylo jasné ve které fázi se nachází. Jakmile je toto nastavení kompletní, je nutné jako poslední určit testovací proceduře proměnnou, která bude definovat hodnotu časování. Většina časů má totiž pevně definovanou dobu trvání a vypršení této doby je de facto další událost, kterou je nutné podchytit ve stavovém diagramu. Proto, pokud uživatel nenadefinuje žádnou takovouto proměnnou, je vyzván aplikací aby tak učinil. Toto konfigurační okno je stejné jako část editoru testovací procedury na obr.2.19 a to v pravém horním rohu tohoto editoru.

Veškeré toto nastavení je nutné definovat již na začátku, když se uživatel přepne do testovací procedury. Aby byla umožněna kompletní editace a úpravy, je možné veškeré toto nastavení následně upravovat v editoru testovací procedury na obr.2.19. Toto okno obsahuje dvě zobrazovací prostředí pro stromové struktury, ve kterém jsou obsaženy nadefinované události(v levém sloupci) a nadefinované vstupy(pravý sloupec). Poslední zobrazovací prvek obsahuje již dříve zmíněné nastavení hodnoty časovače. Dvojitým poklepáním na každý objekt v zobrazených strukturách(kromě kořenových objektů) se vyvolají konfigurační okna zmíněná již výše, v závislosti na zvoleném objektu. Obrázky 2.17 a 2.18 jsou vyjmuté z tohoto nastavení, proto

se v informacích o vyplněných a všech nadefinovaných objektů těchto oken zobrazují informace 1/1, protože se jedná o editaci pouze jednoho objektu ve stavovém diagramu.



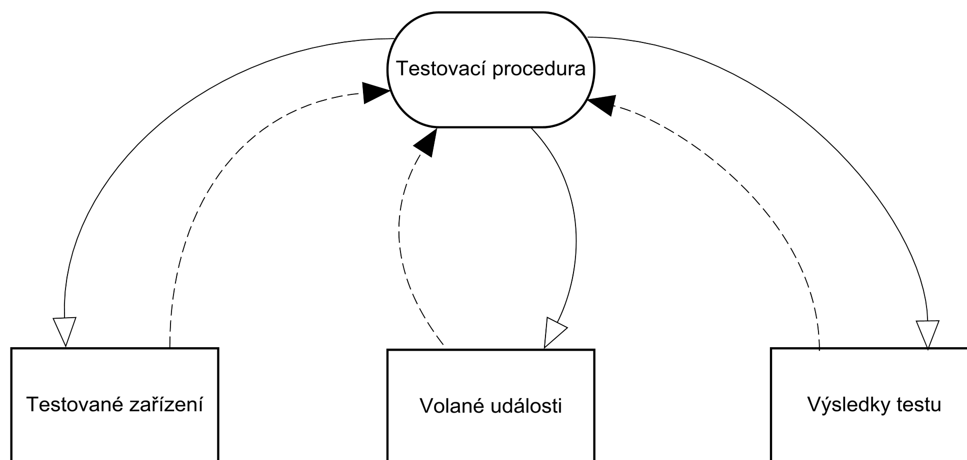
Obr. 2.19: Editor testu

Změna již nadefinovaného objektu je v jádře aplikace provedena jeho úplným smazáním a znovuvytvořením. Tento přístup byl zvolen z důvodu, aby nebylo možné náhodou zaměnit pozice všech konfigurací a funkční proceduru jedinou chybou způsobenou v přístupu k jádru aplikace změnit k horšímu.

2.6.3 Struktura testovací procedury

Na obr.2.20 je naznačena struktura testovací procedury. Prakticky se jedná o výřez z vrstev aplikace a jeho mírně zjednodušená prezentace. Celá procedura je postavena na třech hlavních pilířích, z nichž každý obsahuje nezbytné informace nutné ke správnému testování, ale tyto informace se od sebe liší, takže bylo nutné je od sebe oddělit. Všechny tyto informace využívá testovací procedura, která je vytvořená v hlavním formuláři aplikace. Blok *Testované zařízení* reaguje na události a vrací testovací proceduře odezvy na tyto události. Blok *Volané události* obsahuje veškeré

události, které jsou k dispozici a blok *Výsledky testu* obsahuje strukturu do které se ukládají provedené události a také se zde ukládají informace o pozici ve stavovém diagramu.

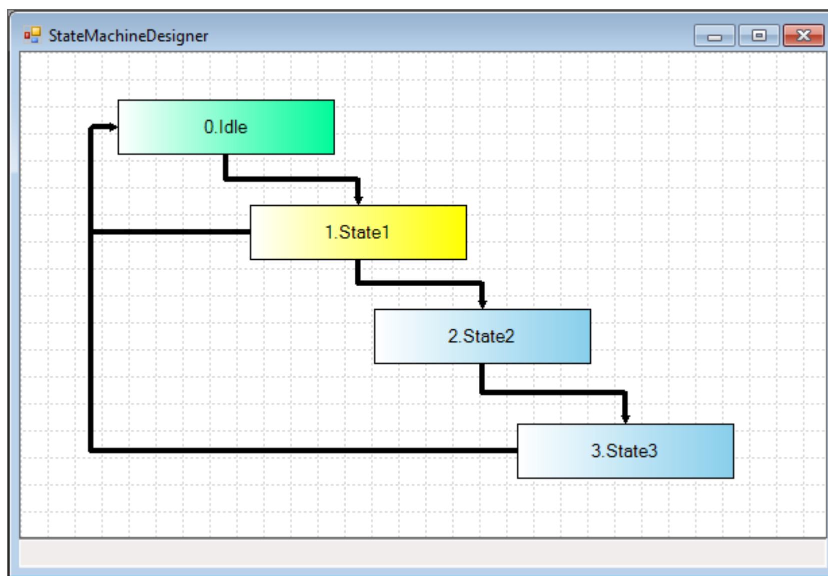


Obr. 2.20: Struktura testování

Veškeré tyto informace jsou nutné ke správnému průběhu testu. Testovací procedura se dotazuje jednotlivých bloků a na základě jejich odpovědí postupuje ve stavovém diagramu. První dva bloky *Testované zařízení* a *Volané události* již byly popsány dříve. Poslední blok obsahuje dvě hlavní struktury. První struktura je seznam stavů, kterými procedura doposud prošla. Tato informace je důležitá pro zjištění, zda se aplikace nezacyklila. Pokud se má testovat tento případ, je zjištěno, kolikrát byl doposud současný stav procházen. Pokud je tato hodnota vyšší, je nutné postupovat nejdelší možnou cestou z tohoto stavu, aby se zamezilo opakování a bylo možné projít veškeré události. Druhá struktura je tvořena slovníkem *Dictionary()* který obsahuje veškeré stavy. Tato informace je tvořena průběhem testu. Dále každý objekt stavu v tomto slovníku obsahuje události které již v tomto stavu proběhly. Srovnáním s jádrem programu takto lze zjistit, které události v určitém stavu nenastaly a je nutné je provést. Také lze zjistit že ve stavu jsou již provedené všechny události. Pomocí této struktury lze také zjistit zda byly provedené všechny stavy a je možné tedy prohlásit že test byl úspěšně provedený. Poslední informací které tento blok obsahuje jsou informace o současném stavu, ve kterém se řidící jednotka nachází a informace o předchozím stavu. Pomocí těchto nástrojů lze provést kompletní test.

Testování tedy v principu funguje tak, že testovací procedura se informuje ve kterém stavu se nachází a zjistí si událost kterou je možné provést. Následně se zjišťuje pomocí bloku *Volané události* informace o události. Není to nic jiného než nalezení události a zjištění názvu potřebné funkce a jejích vstupních parametrů.

Tyto informace jsou zjištěny v jádře a následně je tato funkce zavolána pomocí bloku *Testované zařízení* a na základě odpovědi od zařízení se změní informace v bloku *Výsledky testu*. Všechny tyto bloky obsahují potřebné funkce a blok *Testovací procedura* tyto funkce pouze ve správném pořadí volá a dotazuje se bloků, od kterých je momentálně potřebné získat informace.



Obr. 2.21: Grafická prezentace průběhu testu

Grafická prezentace průběhu testu je naznačena na obr.2.21. Vždy je nejdříve zelenou zobrazeno ve kterém stavu se automat nachází a jakmile se vybere událost která se má provést, stav do kterého tato událost vede se zvýrazní žlutou barvou. Jakmile je událost úspěšně provedena a potvrzen stav kam mělo zařízení přejít tak tento stav je zvýrazněn opět zelenou barvou. Pokud se stav nepotvrdí, je zvýrazněn červenou barvou. Tímto způsobem může uživatel sledovat ve které části stavového diagramu se zařízení nachází, do jakých stavů směřuje a jakmile nastane chyba, je informován o místě kde se tak stalo. V této fázi není možné do testu jakkoliv zasahovat kromě přerušení testu. V tomto případě není graficky prezentováno v editoru že test byl přerušen. Tato informace stejně jako úspěšné ukončení testu a nesprávného výsledku jsou prezentovány pomocí vyskakovacích oken. Pomocí tohoto způsobu jsou dále zobrazovány chyby, pokud se mimo požadovanou sekvenci změní hodnota vstupu. V tomto případě je zobrazeno, který pin na vstupním portu změnil svoji hodnotu. V této fázi je ale pouze prezentována fyzická adresa pinu, ne přímo název.

Je možné tedy říci, že je možné kompletně nakonfigurovat testovací proceduru, spustit a provést test, který prověří že všechny události které definují přechody je možné provést a uživateli je graficky prezentován průběh testu. Struktura testovací

procedury se skládá ze tří hlavních pilířů, které obsahují veškeré potřebné informace se kterými je potřebné pracovat. Konfigurace samotné testovací procedury je relativně zdlouhavá a je nutné znát jak princip testování a způsob nastavení, tak i možnosti knihoven ovládající I/O kartu. V konečném důsledku ale je možné pomocí tohoto způsobu testovat stavové diagramy zařízení.

2.6.4 Vlákno procedury

Aplikace spouští pro samotnou testovací proceduru samostatné vlákno programu. Tato varianta byla zvolena z důvodu řečeného výše - tedy že při samotném testování je možné, že procedura bude čekat například hodinu než se provede požadovaná událost a při tomto čekání by nebylo možné provádět nic jiného a protože je nutné graficky prezentovat postup testu, v hlavním vláknu by nebylo možné vůbec nic zobrazovat ani jej jakkoliv ovládat. Na obr.2.22 je zobrazen podrobnější vývojový diagram vlákna testovací procedury.

Tento diagram je vyvolán po stisknutí tlačítka *Start testing*. Na úplném začátku se vytvoří a spustí nové vlákno v samotné aplikaci. Následné zpracovávání jednotlivých vláken závisí na rozdělení výpočetního času mezi tato vlákna. Díku tomuto je umožněno stále ovládat samotnou aplikaci, graficky zobrazovat průběh testu a při tom stále bude probíhat testovací procedura.

Ihned po vytvoření a spuštění vlákna je v tomto vláknu vytvořena struktura popsaná v kapitole výše na obr.2.20. Jako první je inicializována knihovna pro přístup k I/O modulům. K tomu jsou použité nakonfigurované cesty k souborům popsané v kapitole 2.6.2. Dále je vytvořena struktura kam se ukládají výsledky testu, stavy kterými zařízení procházelo a informace o současné pozici. V této části se také automaticky konfigurují parametry, které mají odstraňovat problém s fyzickými zákmity na vstupním portu. V reálném testu nebude vždy ideálně ihned sepnutý spínač, u elektromagnetických cívek také dochází k zákmitům a tyto zákmity informují aplikaci o nežádoucích změnách již nastavené hodnoty a způsobí tak nesprávnou interpretaci akcí. Tento efekt je odstraněn tím, že knihovna pro I/O moduly si nastaví časy, po kterých se určitá logická hodnota považuje za ustálenou.

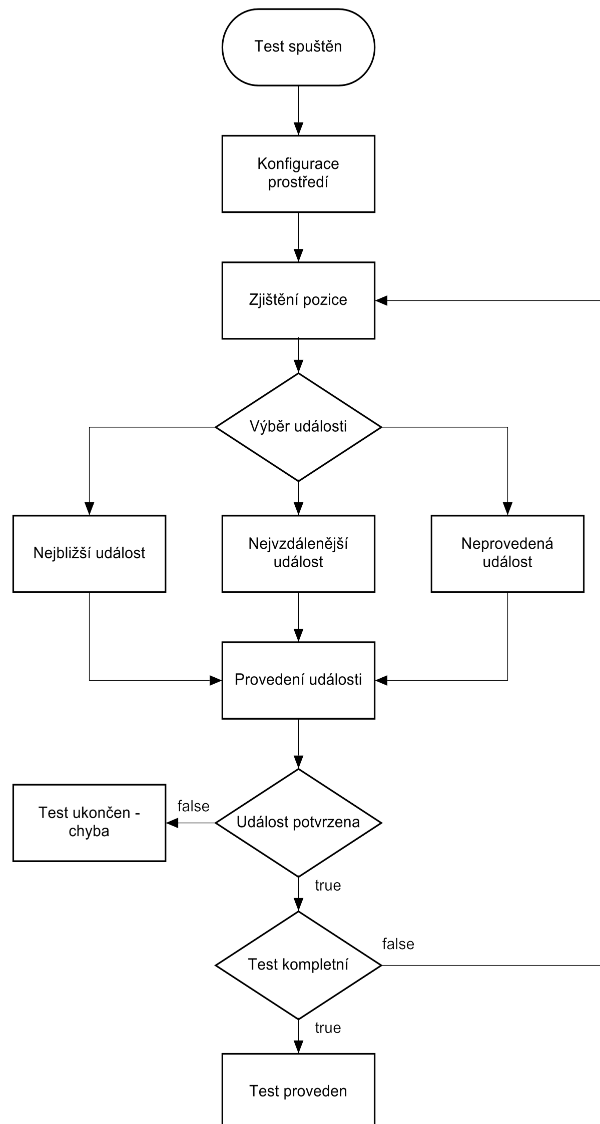
V dalším kroku se již pohybujeme ve smyčce, která má nastavenou hodnotu tak, že po provedení kompletního testu, nebo při přerušení od uživatele, nebo při chybě v testování je z tato smyčka opuštěna a vlákno je tímto způsobem ukončeno. Tato smyčka začíná zjištěním aktuální pozice ve stavovém diagramu. Blok, ve kterém jsou uloženy výsledky a aktuální pozice obsahuje, jak bylo již výše řečeno, proměnné nesoucí informaci o současném stavu a o předchozím stavu. Na začátku testu jsou obě tyto proměnné prázdné. Tímto způsobem testovací procedura zjistí, že je na začátku testu a je potřeba vyhledat první stav ve stavovém diagramu. Když je známá pozice,

ve kterém se testovaná jednotka nachází, následuje výběr události. Způsob výběru již byl popsán výše. Testovací procedura na tomto základě vybere událost a jakmile je znám název události, kterou je nutné provést, testovací procedura zjistí informace o této události v jádře aplikace. Jedná se o název volané funkce, vstupní parametry a typ těchto parametrů. Na základě informací je tato událost vyvolaná.

Výše v práci již bylo řečeno, že testování v nejzákladnějším smyslu je pouze sekvence akcí a očekávání určitých reakcí na ně. Proto aby bylo možné v další kroku vlákna zjistit výsledek, že událost vedla do správného stavu, je nutné očekávat do určité doby změnu informace na vstupním portu. Každý stav je totiž definovaný kombinací logických jedniček a nul na vstupním portu. Tato definice vyplývá ze struktury stavového diagramu, který je uložen v jádře. Zde je ale nutné si uvědomit, že uživatel nastavuje v návrhu stavového diagramu pouze ty vstupy, které jsou pro daný stav aktivní. Z tohoto důvodu si musí aplikace sestavit vstupní port ze všech názvů které má definované v celém stavovém diagramu a změnit pouze ty názvy do logických jedniček, které jsou definovány pro stav který je očekáván.

Pro kontrolu, zda port dosahuje potřebných hodnot slouží funkce, kterou obsahuje knihovna I/O modulů. Tato funkce čeká na kombinaci která je jí předložena po určitý čas, který je nutné také definovat. Výstupem z funkce je hodnota času, kdy se tento vstupní port dostal do požadované hodnoty. Pokud je překročen definovaný limit, funkce vrátí hodnotu -1. Tento limit je definovaný proměnnou kterou uživatel definoval jako *Timer*. Na základě výsledku se poté vyhodnocuje výsledek. Během této kontroly, stejně jako během celého testu je vstupní port uzamčený a jakmile se změní jeho hodnota během testování, test je vyhodnocen jako neúspěšný a je také ukončen. Toto je ošetřeno z toho důvodu, že některý vstup se může změnit, ale ve stavovém diagramu nejsou dovolené žádné odchylky. Taková odchylka znamená, že je chyba v zařízení nebo v samotném návrhu stavového diagramu. Tato funkce je do testovací procedury implementována přímo a uživatel k ní nemá žádné přístupy, kromě nastavení hodnoty časovače. Tento způsob byl zvolen výhradně proto, že v této práci se testují reakce na úrovni digitálních hodnot, ne analogových. V průběhu celého testování se nevyskytla nutnost implementace jiných způsobů čtení reakcí testovaného zařízení, při dalším rozšiřování testovací aplikace by ale bylo vhodné takovouto možnost do testovací procedury implementovat, protože samotná aplikace umožňuje uživateli nastavovat jakékoliv myslitelné funkce, které jsou dostupné v knihovně I/O modulů.

Pokud není stav potvrzen, test je ukončen jako chybný. Jakmile je stav potvrzen, pokračuje se dále v testovací proceduře. Poslední akce, která se na konci smyčky provádí je kontrola, zda je test kompletní. Tato akce spočívá v porovnání úspěšně provedených událostí a celkovému číslu všech událostí obsažených ve stavovém diagramu. Jakmile jsou události provedené všechny, test je ukončen jako úspěšně



Obr. 2.22: Vývojový diagram průběhu testování

provedený a navržený stavový diagram odpovídá reálnému řidícímu automatu testovaného zařízení. Pokud nejsou provedené všechny události, smyčka není přerušena a znovu se vrací na začátek, kde je znovu zjištěna aktuální pozice a vybrána další událost. Celá smyčka je řízena proměnnou *testPassed*, která tuto smyčku ukončuje v kterémkoliv ze čtyř případů které mohou nastat - správně provedeném testu, chybě při testování, přerušení uživatelem a změnou vstupního portu když není tato změna očekávána. Identifikace chyby se následně děje společně s touto akcí.

Pokud je stav, do kterého se musí přejít identifikovatelný stejnou kombinací vstupů jako předchozí stav, událost je prakticky okamžitě potvrzena i přes fakt, že se ve stavu nemusí řidící jednotka nacházet. Tato informace by byla potvrzena kdybychom znali vnitřní informace o řidící jednotce. Protože ale testování v této

práci se zabývá pouze testem 'Black box', je nutné počítat s neurčitostí v testování, kdy není přesně jisté zda se zařízení opravdu v tomto stavu nachází. Následný postup ve stavovém automatu a také následné chyby v definovaných přechodech umožňují následnou identifikaci.

2.6.5 Zapisování a prezentace výsledků

Vyhodnocení výsledků testu pouze informací zda byl test proveden kompletní, nebo zda byl neproveden je velice obecná informace a neumožňuje zkontrolovat veškeré parametry, které se při testování stavových automatů mají prověřit. Některé podrobnosti lze obsáhnout přímo v testovací aplikaci. Z tohoto důvodu se veškeré časování, které je definováno ve stavovém automatu navyšuje automaticky o deset procent. Tímto se podchycují nepřesnosti, které mohou vzniknout v samotném procesoru testovaného zařízení, ale také přímo v operačním systému počítače, na kterém je aplikace spuštěná. Mezi další parametry ale patří zda byly všechny přechody ve stavovém automatu provedené tak jak měly být. Tuto informaci ale nelze získat až na samotném konci testu, je nutné veškeré události zaznamenávat jako určitý soubor nebo objekt, který bude následně uživatelem použit pro sumarizaci výsledku testu.

Ve vytvořené aplikaci se data o průběhu testu automaticky zapisují do textového souboru, který je uložen ve složce *Results*. Takovýto soubor je automaticky vytvořen vždy na začátku každého testu. Pro zjednodušení práce s výsledky aplikace vytváří soubory takovým způsobem, aby uživatel nemusel každý soubor popisovat již od vytvoření. Je to tak provedeno z toho důvodu, že samotná aplikace již vyžaduje velkou míru nastavení, aby bylo možné takovýto test provést a každá takováto akce zbytečně prodlužuje čas, který uživatel stráví editací a sumarizováním výsledků testu. Existují totiž možnosti, že lze soubor do kterého se zapisují výsledky pojmenovat jednoduchým názvem (kupříkladu *result*), mít jej vytvořený pouze jednou a vždy při každém spuštění testu smazat jeho obsah a znovu do něj zapisovat. Tento způsob ale pouze předpokládá, že uživatel vždy tento soubor překopíruje do své vlastní složky a následně jej přejmenuje dle svého uvážení. V praxi se ale velice často stává, že uživatel spouští testy a vždy při chybě testování testovací proceduru pouze mírně upraví a pokračuje v dalším testování. Tímto způsobem ale uživatel přichází o data z testování, i když jsou tato data chybná. Je možné že později i tato data budou potřebná a informace o těchto souborech bude ztracená.

Z tohoto důvodu je pro každý takový spuštěný test vytvořený samostatný textový soubor ve složce *Results* přímo v souborech aplikace, kde již jsou uložené vytvořené projekty a XML soubory. Aby nedocházelo ke konfliktům, každý soubor je uložený pod relativně delším názvem. Tento název obsahuje informace, které je nutné zadat vždy na začátku každého nového projektu - tedy jméno uživatele, který projekt

vytváří a název testovaného zařízení. Jak již ale bylo zmíněno výše, pokud uživatel tato pole nevytvoří, jsou automaticky doplněna předdefinovaným textem 'Unknown'. Takovýto způsob nastavování neřeší problém se stejným názvem souboru. Proto byl do názvu textového souboru přidán další identifikační parametr a tím je datum, rok a čas, kdy byl test spuštěn a to s přesností na vteřiny, protože je prakticky nemožné spustit tentýž test ve stejný okamžik. Díky tomuto je minimalizována možnost, že informace o průběhu testu bude nějakým způsobem ztracena. Samozřejmě že tím narůstá počet vytvořených souborů ve složkách aplikace a takovýchto textových souborů je vytvořeno velice mnoho, ale tímto způsobem se neztratí žádná důležitá informace a každý soubor je také dostatečně identifikovatelný. Nelze ale zamezit tomu, že uživatel soubory smaže, ale takovýto zásah nelze ovlivnit. Nedojde již ale k přepsání souboru a uživatel nemusí vyplňovat název souboru ručně.

Samotný textový soubor obsahuje nejprve hlavičku, která obsahuje přesněji pojmenované parametry, kterými je pojmenován soubor, legendu která popisuje strukturu záznamu, samotné záznamy o přechodech a výsledek testu. Na obr.2.23 je příklad takovéto hlavičky společně s legendou k zápisu informací o přechodech. Lze zde vidět kdy přesně byl test spuštěn, k jakému zařízení se vztahuje a také obsahuje jméno uživatele. Informace obsažená v tomto zápisu je kompletní a není nutné přidávat další informace. Legenda je pro každý textový soubor totožná. Popisuje v zásadě to, že na prvním místě je ze kterého stavu se vycházelo při spuštění události, za šipkou následuje stav do kterého zařízení přešlo, v závorce je uvedena samotná událost, která vedla k tomuto spuštění a za dvojtečkou následuje doba, za kterou bylo tohoto přechodu dosaženo.

```
Test started 4/21/2015 9:06:23 AM
Device: SV9501
User: Stanislav Horky

Legend: From State -> Next State (Event that caused transition) : Duration of transition
```

Obr. 2.23: Hlavička záznamu testu

Pomocí tohoto záznamu může uživatel dohledat a přesně zjistit chování stavového automatu testovaného zařízení. Lze zde dohledat přesný čas, který je uváděn v milisekundách a veškeré informace vztahující se k testovanému přechodu.

Další část textového souboru již obsahuje samotné záznamy o přechodech. Protože testy probíhají v cyklech, je pro oddělení takovýchto cyklů použit nejprve záznam *Test run* s číslem cyklu. Jakmile se zařízení vrátí do výchozího prvního stavu, je hodnota počtu provedených cyklů inkrementována a znovu oddělena v záznamu. Vytvoření každého záznamu o provedeném přechodu je provedeno pouze v případě, že přechod byl úspěšně proveden a jsou o něm známé veškeré informace, které je

nutné zapsat. Jakmile dojde k překročení definovaného času, je do potřebného pole, kam se zapisuje uložený čas zapsán text *Timeout*. Tímto je možné uživatele informovat o tom, že takovýto určitý přechod způsobil neúspěch celého testu. Na obr.2.24 je příklad části takového záznamu. Zároveň je také každý přechod zaznamenán společně s časem, kdy byl tento přechod úspěšně proveden. tato informace je zde pro úplnost, aby bylo možné se následně odkazovat na určitý řádek v záznamu. Veškeré obrázky vztahující se k záznamu v textovém souboru jsou převzaté přímo z praktického testu, který bude popsán v kapitole vztahující se k tomuto tématu.

```
Test run 0
06:57:31 Idle->Trial For Ignition 1(Power On): 6512
06:57:34 Trial For Ignition 1->Running(Flame ON): 2192
06:57:38 Running->Interpurge(Flame Off): 4070
06:57:45 Interpurge->Trial For Ignition 1(Timer Expires 1): 4061
```

Obr. 2.24: Záznamu testu

Na konci záznamu je uveden obecný výsledek celého testu. Úspěšně provedený test je označen jako *Passed*, neúspěšný test je označen jako *Failed* a v tomto druhém případě, pokud nastane chyba vyvolaná nesprávnou změnou na vstupním portu, tato chyba je zapsána také na konec testu. Je možné tedy prohlásit, že uživatel má po provedení testu k dispozici úplnou informaci o podrobném průběhu a s tímto faktem lze dále pracovat.

Celý tento popis záznamu je kompletně integrován do samotné testovací procedury na příslušných místech a po skončení testu je vytvořený textový soubor o záznamu uzavřen, stejně jako samotné testovací vlákno a také jsou uzavřeny přístupy ke knihovnám pro I/O soubory.

Zápis výsledků je tedy pro každý test proveden do zvláštního textového souboru, který uživateli umožňuje prezentovat výsledky testu dále. Jak již bylo popsáno výše, aplikace umožňuje i grafickou prezentaci o průběhu testu v reálném čase spolu s předpokládaným dalším postupem ve stavovém diagramu a také se zobrazením úspěšně provedených přechodů zvýrazněním příslušných stavů, jakožto i zobrazení neúspěšně proběhlých přechodů, také zvýrazněním příslušných stavů. Je tedy možné říci, že vzhledem k výsledku testu je uživatel informován v průběhu testování a na konci má podrobný výpis, který je pro další práci dostačující.

2.6.6 Modifikace aplikace

Uživateli je umožněno pomocí aplikace provést kompletní test stavového automatu. Navrženou proceduru by bylo možné ale dále modifikovat, což je směr kterým by se měla práce na aplikaci dále ubírat. Jednou z důležitých modifikací by mohlo být

opuštění testování "Black box"zařízení, kdy nejsou známé informace o vnitřních procesech. Jak bylo již řečeno výše, v určitých fází není přesně jisté, zda se zařízení v určitém stavu nachází, jelikož se jeho chování nemusí změnit. Tímto příkladem můžou být jakési 'virtuální' stavy - tedy stavy, které není možné přesně identifikovat na základě chování samotné jednotky. Pro vyřešení tohoto problému by bylo například možné, aby vytvořená testovací aplikace dokázala pracovat spolu se sériovým portem, kterým by mohlo testované zařízení posílat informace o tom v jakém stavu se nachází. V praxi zařízení mají diagnostické výstupy, nebo alespoň umožňují komunikaci po UART. Samotná problematika nastavení a čtení informací po sériovém portu je velice obsáhlá a již by se svým rozsahem nevešla do této práce. Dále by mohla testovací procedura umožňovat komunikaci po jiných typech komunikačních standardů, například USB, I2C a podobně.

Další modifikací by mohlo být nastavování analogových výstupů definovanými hodnotami, kdy by bylo možné následně přesněji otestovat chování testované jednotky.Celá problematika automatizovaného testování nabízí širokou paletu možností, které by bylo možné implementovat do vytvořené aplikace, kdy by například bylo vytvořeno více testovacích procedur, z nichž by každá testovala různým způsobem. V této fázi se testují veškeré přechody. V další by se mohl vytvářet přesněji definovaný skript, který by se mohl prakticky skládat z obecně definovaných modulů a tak otestovat že se zařízení chová korektně i při aktivních řídících impulzech, které nejsou v určitém stavu žádoucí.

Bylo zde zmíněno, že aplikace nastavuje fixní doby, po kterých je vstup považován za ustálený. V tomto případě by bylo výhodnější, kdyby tyto parametry nastavoval přímo uživatel a bylo by možné je modifikovat.

Samotné zapisování výsledků by bylo pak následně potřeba modifikovat, aby bylo možné například i zaznamenávat informace posílané po sériové lince. U grafické prezentace průběhu testu je možné uvažovat o tom, že by se zobrazovaly také přechody ve stavovém diagramu, které již byly provedené aby bylo možné ihned vizuálně zjistit, který přechod ještě nebyl proveden. K vizuálnímu grafickému editoru by bylo také možné přidat konzolové okno, které by informovalo o inicializaci knihovny pro I/O moduly a bylo by možné zde sledovat a zachytávat chyby, které se mohou projevit mimo hlavní část aplikace, tedy v již zmíněných knihovnách I/O modulů. Celá aplikace by pak následně připomínala jakési vývojové prostředí.

3 TEST APLIKACE

V rámci vytvoření aplikace bylo nutné prakticky ověřit její správnou funkčnost. Pro tento případ byla otestována dvě zařízení. V této kapitole jsou podrobněji popsané principy funkčnosti těchto zařízení, jednotlivá zapojení a jejich následné ověření pomocí testovací aplikace.

3.1 STK500 s procesorem ATmega16

Prvním testovaným zařízením je procesor ATmega16 připojený na vývojovou desku STK500, která umožňuje programování, oživování a další vývojové nástroje aniž by bylo nutné vytvářet podpůrnou desku s napájením pro vybraný mikroprocesor. Výběr procesoru byl čistě náhodný a slouží pouze pro demonstraci funkčnosti. Podrobné parametry vývojové desky a použitého mikroprocesoru zde není nutné uvádět, jelikož nejsou pro samotný test důležité.

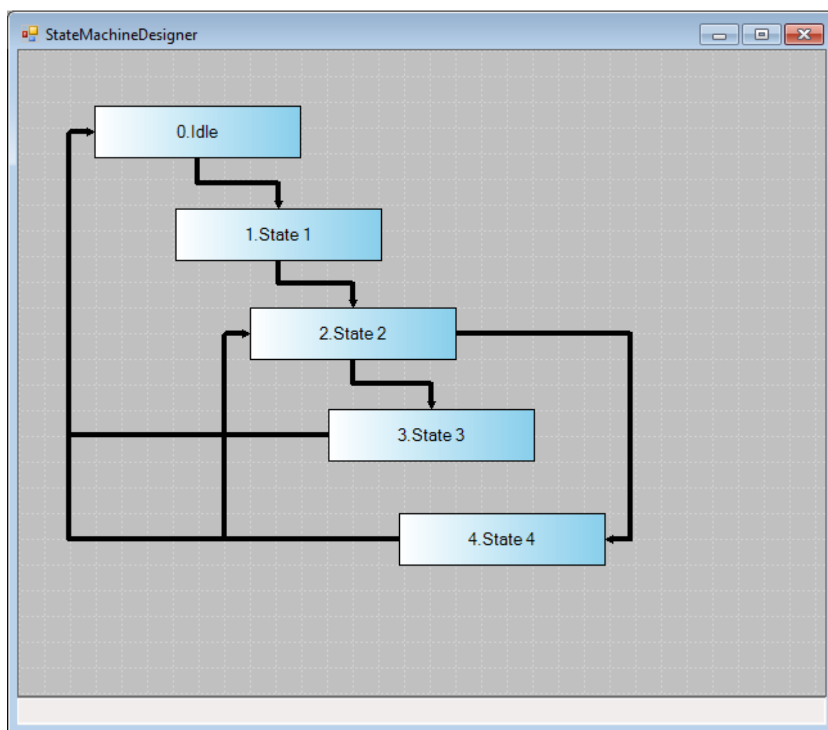
3.1.1 Vytvořený projekt

Program vytvořený v procesoru ATmega16 je jednoduchý a jeho funkce je čistě jen taková, že kombinaci logických úrovní na vstupním portu invertuje a vrátí na výstupní port jejich obrácenou hodnotu a celá tato akce je provedena v rozmezí jedné a půl vteřiny. Tímto lze navrhnout prakticky libovolný stavový automat, na kterém se ověří, že aplikace prochází stavový diagram takovým způsobem, jakým byla navržena. V závislosti na počtu vstupních pinů a počtu výstupních pinů lze vytvořit určitý počet stavů ve stavovém automatu. Pro dva vstupní a dva výstupní piny platí kombinace čtyř stavů, tato verze byla použita při testování. Při implementaci třech pinů lze vytvořit osm stavů, což již umožňuje navrhnout relativně rozsáhlý stavový automat, který se přibližuje svým rozsahem jednoduchým stavovým automatům reálných zařízení. Do programu byla následně implementována tlačítka, fyzicky umístěná na vývojové desce STK500, jejichž stiskem lze narušit stavový automat, tím že příslušný pin náležící tlačítku bude po dobu stisku tlačítka neustále v logické jedničce. V tomto případě nebude aplikaci souhlasit vstupní port odezvy od zařízení a program zahlásí chybu.

Na obr.3.1 je zobrazen vytvořený stavový diagram, který byl aplikací otestován. Aby bylo možné otestovat správnou funkčnost navrženého testovacího algoritmu, byly do navrženého stavového diagramu záměrně vloženy stavy tak, aby obsahoval alespoň jednu odbočku z hlavní smyčky. Tato odbočka je realizována stavem s názvem *State 4*. Po průchodu hlavní smyčky je nutné aby se algoritmus dostal po průchodu hlavní smyčkou do stavu číslo dvě a následně přešel do stavu číslo čtyři.

Dle stavového diagramu z tohoto stavu vedou další dva přechody - jeden zpět do stavu dvě a druhý do prvního stavu. Jakmile zařízení provede jeden z těchto přechodů, je nutné aby se znovu dostalo do stavu číslo čtyři a byl otestován poslední přechod. Je velice pravděpodobné, že v tomto případě algoritmus bude určitou dobu cyklovat v hlavní smyčce, dokud se nedostane znovu do stavu číslo čtyři a otestuje poslední přechod. Poté bude test úspěšně provedený.

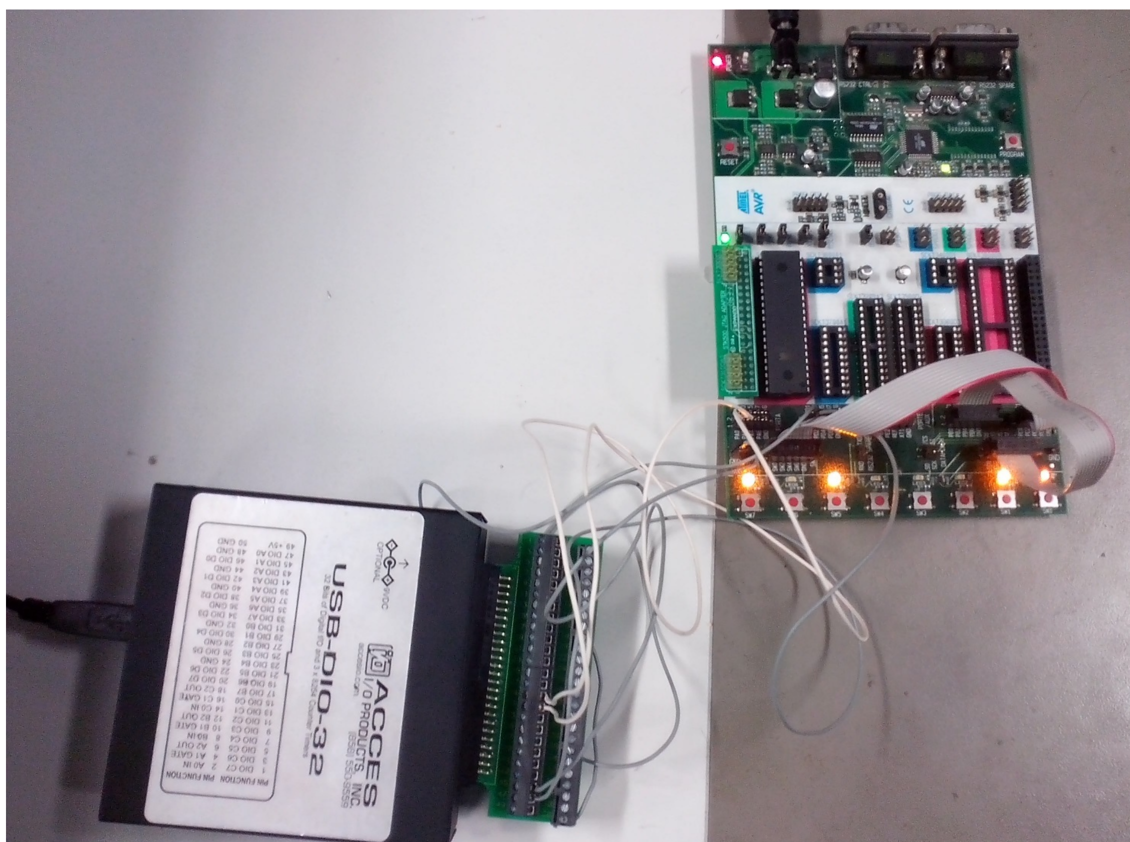
Pro zařízení byly definovány dva testy. U prvního, který je výše popsán, je předpoklad že bude úspěšný. Druhý test má otestovat funkčnost tlačítek, respektive funkci že lze s jejich pomocí vyvolat chybové hlášení. V určité fázi testu je stisknuto jedno z tlačítek a zařízení je vnucena nesprávná hodnota a tím pádem je tato hodnota vnucena i testovací proceduře a ta by následně měla vyhodnotit chybu a v tomto smyslu bude také ve výsledném záznamu negativní průběh testu spolu s informací v které fázi tato událost nastala.



Obr. 3.1: Diagram řídícího automatu pro zařízení s vývojovou deskou STK500

Na obr.3.2 je fotografie pracovní plochy se zapojeným zařízením STK500. Na levé straně je karta ACCESS USB-DIO-32, která slouží ke komunikaci mezi testovací procedurou a aplikací. Tato karta využívá 50-ti pinový konektor, který obsahuje také interní čítače obsažené na kartě. V důsledku toho karta poskytuje možnost pro připojení na čtyři osmipinové porty z nichž každý může být nadefinovaný jako vstupní nebo výstupní. Toto všechno je nutné nastavit v konfiguračním souboru aby mohla aplikace komunikovat s touto I/O kartou. Karta je dále připojena přes

rozhraní USB do počítače s testovací aplikací. Karta není připojena na plochý kabel, ale na redukci na kterou je možné připojit jednotlivé vodiče pomocí svorkovnic. Celé zapojení je velice jednoduché, jelikož není potřeba žádných převodníků, protože výstup a vstup do vývojového kitu STK500 je uzpůsoben na TTL logiku, stejně jako I/O karta.



Obr. 3.2: Fotografie pracoviště s vývojovou deskou STK500

Aby bylo možné sledovat funkčnost samotného zařízení STK500 s procesorem ATmega16,hodnoty výstupního portu, který reprezentuje odpovědi od zařízení do testovací aplikace, jsou vyvedeny navíc ještě na LED diody umístěné přímo na vývojovém kitu. Takto je možné pozorovat změny, které jsou prováděny a následně je prověřit i v testovací proceduře. Oba testy byly provedeny a výsledky těchto testů jsou uvedeny v následující kapitole.

3.1.2 Výsledky testů

Pro potřeby ověření samotné funkčnosti byly provedeny dva testy popsané v kapitole výše. Výsledky prvního testu jsou na obr.3.3. Je to vlastně kompletní záznam z textového souboru. Hlavička souboru obsahuje informace o zařízení, jméno uživatele a legendu. Pro kompletní otestování navrženého diagramu testovací procedura

potřebovala tři testovací cykly. Tolikrát procedura prošla výchozím prvním stavem stavového diagramu.

Jak lze vidět, test proběhl úspěšně, byly otestovány veškeré přechody ve stavovém diagramu. Každý přechod trval přibližně tři vteřiny, což mohl způsobit debouncing provedený na DIO kartě nebo algoritmus v procesoru ATmega16, na tento fakt ale není zaměřena tato práce. Samotné měření času v testovací aplikaci je sice měřeno na jednotky milisekund, ale přesnost takovéhoto měření není příliš veliká, pro přesnější měření by bylo nutné použít přesnější časovač, na který existuje více projektů dostupných na internetu. Pro potřeby softwarového testování je toto ale dostačující, důležitější bude přesnost u samotného reálného výrobku.

Test byl tedy úspěšný a bylo ověřeno, že navržená testovací procedura je schopná si poradit i s větvením programu a dalšími problémy které již byly výše v práci popsány.

```
Test started 5/5/2015 8:08:20 AM
Device: STK500
User: Stanislav Horky

Legend: From State -> Next State (Event that caused transition) : Duration of transition

Test run 0
08:08:22 Idle->State 1(Power On): 2399
08:08:25 State 1->State 2(Event On): 3086
08:08:29 State 2->State 3(Event Off): 3118
08:08:32 State 3->Idle(Power Off): 3118

Test run 1
08:08:35 Idle->State 1(Power On): 3096
08:08:38 State 1->State 2(Event On): 3128
08:08:41 State 2->State 4(Power Off): 3084
08:08:44 State 4->Idle(Event Off): 3102

Test run 2
08:08:47 Idle->State 1(Power On): 3118
08:08:51 State 1->State 2(Event On): 3119
08:08:54 State 2->State 4(Power Off): 3087
08:08:57 State 4->State 2(Power On): 3118

Passed
```

Obr. 3.3: Výsledek prvního testu stavového automatu zařízení STK500

Na obr.3.4 je provedený druhý test na zařízení STK500 s procesorem ATmega16. V průběhu stejného testu jako v prvním případě bylo stisknuto jedno tlačítko, které podrželo jeden výstup zařízení v aktivní jedničce a tím bylo dosaženo narušení testu. Výsledek testu je tedy takový, že test nebyl dokončen, ale to je v tomto případě žádoucí, protože tento fakt říká, že zařízení je opravdu schopné reagovat na chybová hlášení od zařízení, které mohou vzniknout špatným návrhem diagramu nebo chybou v řídícím automatu testovaného zařízení.

V případě tohoto testu nastalo chybové hlášení ve druhém testovacím cyklu. Časy všech přechodů jsou podobné výsledkům z prvního testu a postup procedury je identický do doby stisknutí tlačítka a vyvolání chybového hlášení. Ze záznamu lze vyvodit, že chyba nastala v přechodu ze stavu číslo jedna do stavu číslo dvě při pokusu o vyvolání reakce na událost Event On a důvodem, proč test neprošel bylo vypršení limitu ve kterém se událost měla provést. Pokud se ale zaměříme na výstup ze zařízení, který měl zůstat v logické jedničce, z testu vyplývá, že tomu bylo přesně naopak. Z posledního záznamu lze vyvodit, že výstup se měl změnit na aktivní, ale zůstal neaktivní a proto celý test havaroval. Tento fakt je způsobený tím, že celý výstup ze zařízení je samotným programem v tomto zařízení invertován na výstup. Pokud tedy uživatel stiskne tlačítko a tím se určitý vstup přepne do logické jedničky, výstup zařízení je invertován, je to tedy v konečném důsledku logická nula.

```
Test started 5/5/2015 8:13:56 AM
Device: STK500
User: Stanislav Horky

Legend: From State -> Next State (Event that caused transition) : Duration of transition

Test run 0
08:14:00 Idle->State 1(Power On): 3501
08:14:03 State 1->State 2(Event On): 3129
08:14:06 State 2->State 3(Event Off): 3097
08:14:09 State 3->Idle(Power Off): 3128

Test run 1
08:14:11 Idle->State 1(Power On): 1538
08:14:28 State 1->State 2(Event On): Timeout

Failed
```

Obr. 3.4: Výsledek druhého testu stavového automatu zařízení STK500

Pomocí těchto dvou testů byla tedy otestována funkčnost celé testovací procedury. Samotný program v procesoru ATmega16 je v tomto případě použitý pouze pro ověření a není příliš sofistikovaný. Bylo ale s jeho pomocí ověřeno, že aplikace a testovací procedura v ní vytvořená je funkční a schopná testovat stavové automaty připojených zařízení. Přiložené CD obsahuje video z obou těchto testů na kterých je vidět samotná grafická prezentace v prostředí aplikace a také reakce samotného zařízení spolu s reakcemi prezentovanými pomocí LED diod na vývojové desce STK500.

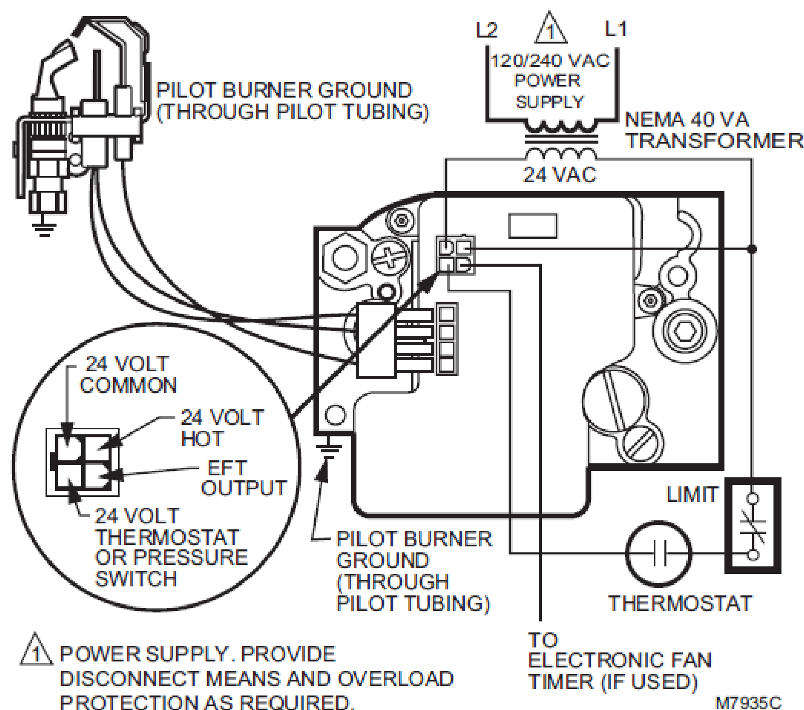
3.2 Smart Valve SV9501

V zadání této práce je uvedeno že funkčnost aplikace se má demonstrovat alespoň na jednom projektu ve vývojovém centru firmy Honeywell. Pro tento příklad bylo

vybráno zařízení Smart Valve. Jedná se o jednotku, která řídí samotný plynový ventil a je přímo osazena na těle tohoto ventilu.

Jedná o zařízení používaného v aplikacích používající ohřev plynem. Tyto aplikace zahrnují například centrální pece, domácí kotle a komerční kuchyňské spotřebiče nebo ohřívače. Poskytují sekvenci zapalování pomocí pilotního ventilu, měření plamene a ovládací funkce pro pilotní a hlavní ventil, to vše v jedné jednotce.[9]

Řada jednotek Smart Valve nabízí vícero typů, které si může zákazník podle své potřeby vybrat. Pro tuto práci byl zvolen přesný typ Smart Valve SV9501M2528. Rozdíly mezi jednotlivými typy je pouze v časování jednotlivých úkonů a velikosti ovládaných ventilů, nebylo tedy třeba volit zvláštní typ. Pro otestování aplikace by bylo možné použít prakticky jakékoliv zařízení. Na obr.3.5 je obecné schéma zapojení převzaté z instalačních instrukcí[9]. Podle tohoto zapojení byla vytvořena testovací fixtura pro samotné zařízení.



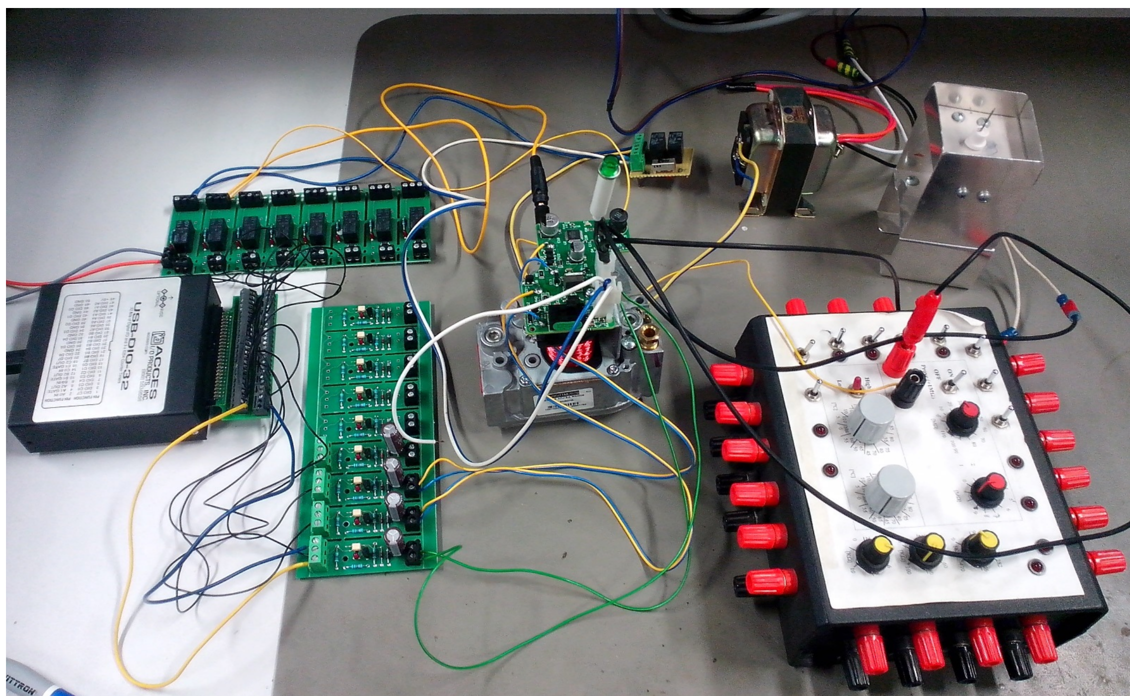
Obr. 3.5: Schéma zapojení z instalačních instrukcí[9]

Proto aby bylo možné úspěšně otestovat celé zařízení, je nutné jej zapojit přesně podle uvedeného schématu. Protože testovaná jednotka je ve své podstatě plynový ventil, není možné jej testovat přímo na pracovní ploše, protože by v blízkosti uživatele byla přítomnost plynu a samozřejmě také přítomnost samotného hořícího plamene. Nehledě na nutnost odvětrávání a dalších přídavných zařízení, která jsou

přítomna v samotném kotli určeném k vytápění. Proto jsou některé tyto věci simulovány vhodnými prostředky. Zapojení testovacího pracoviště je na obr.3.6. Přesné zapojení zde nebude potřebné uvádět, je třeba pouze vysvětlit princip celé fixtury v závislosti na obr.3.5.

Na levé straně pracoviště je již výše zmíněná I/O karta USB-DIO-32. Protože ale testované zařízení není postaveno na principu pěti-voltové logiky vstupů a výstupů, je nutné použít oddělovací obvody. V případě vstupů do I/O karty se jedná o to, že výstup lze považovat za aktivní, pokud je na něm přítomno 24VAC, tedy střídavé napětí.Je tedy nutné toto napětí filtrovat a usměrňovat dostačujícím kondenzátorem a následně přes optické oddělení dodávat dále, kde je již přítomna TTL logika. V případě výstupů z I/O karty je nutné si uvědomit, že karta dodává TTL logiku, ale úkolem obvodů je pouze simulovat spínač zapojený do obvodu. Z tohoto důvodu je tímto výstupem ovládáno relé, které sepne příslušný kontakt.

V pravém dolním rohu pracoviště je modul, který slouží pro simulaci plamene. V pravém horním rohu je v plechovém stojanu umístěno zapalovací zařízení, neboli Hot Surface Ignitor, což je jediná reálná věc, kromě samotné řidící jednotky, která není simulována.



Obr. 3.6: Fotografie pracoviště s SV9501

Uživatel, nebo spíše testovací aplikace, bude mít přístup pouze ke dvou ovládacím prvkům. Prvním bude připojení napájení k jednotce, čímž se spustí zapalovací sekvence a druhým bude simulování prezence samotného plamene. Pro základní

otestování jednotky je toto dostačující. Zařízení dále obsahuje spínač kontrolující vysokou teplotu(Limit), ale v tomto případě bude neustále sepnutý.

Výstupem zařízení a tímto tedy i detekce ve kterém stavu se zařízení nachází, budou tři výstupy. Prvním z nich je spuštění zařízení Hot Surface Ignitor. To je prezentováno napětím 24VAC na konektoru vedoucím přímo na toto zařízení. Dalšími dvěma výstupy bude pilotní a hlavní ventil, respektive zda jsou otevřené nebo zavřené. Přítomnost 24VAC na svorkách které ovládají elektromagnety k ventilům znamenají, že ventil je otevřený a pokud toto napětí není přítomno, ventil je zavřený. Bylo by možné pro přesnější detekci použít stlačený vzduch a senzory tlaku, což ale není v tomto případě, kdy je nutné pouze otestovat základní funkčnost samotné desky, potřebné.

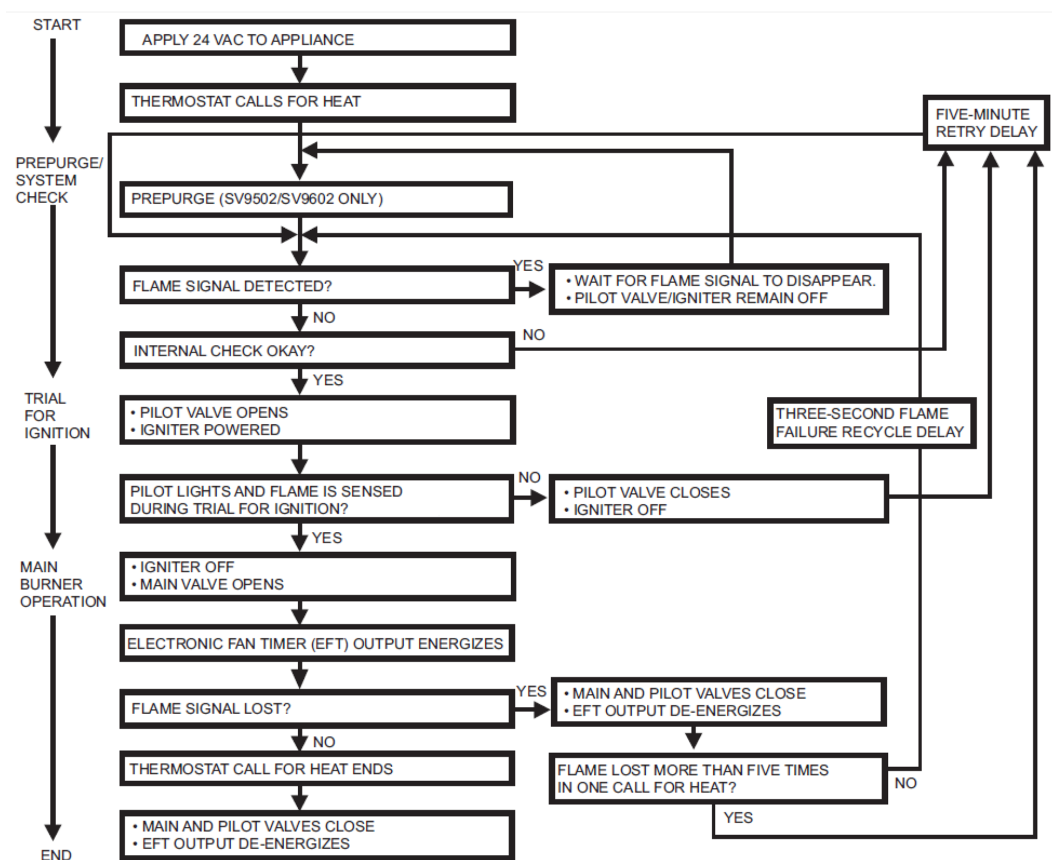
Na samotném zařízení je ještě přítomný EFT(Electronic Fan Timer) výstup, který vede k dalšímu externímu zařízení. Tento výstup nebude využit protože není nutný pro správnou funkčnost samotné řidící jednotky. Celé zapojení bylo koncipováno tedy tak aby bylo co nejjednodušší, ale zároveň aby byla umožněna správná funkce řidící jednotky. Celá testovací fixtura je napájená ze sítě 120V/60Hz.

3.2.1 Vytvořený projekt

Všechny vytvořené projekty pro testování zařízení Smart Valve se vztahují ke stavovému automatu popsaného na obr.3.7. Z tohoto diagramu se vycházelo při tvoření projektů. Když chce uživatel vytvořit takovýto projekt přímo pro určené zařízení, je nejprve nutné analyzovat stavový diagram zařízením jak jej popisuje výrobce. Diagram na obr.3.7 je na první pohled složitý, ale po zjednodušení jej lze jednoduše otestovat.

V první řadě je třeba určit stavy zařízení. Zde je možné vycházet z levé strany tohoto diagramu. Po zapnutí napájení pro zařízení je aktivován požadavek na topení. Následuje kontrola systému a poté mají určité verze zařízení definovaný stav *Prepurge*, což znamená příprava na zapálení plamene. Testovaná jednotka v tomto dokumentu je typu SV9501 a tento stav se jí netýká. Celý tento proces lze sjednotit pod jeden jediný stav, který zde bude nazývaný *Idle*, tedy výchozí stav. V průběhu celého stavu je možné přejít do takzvaných chybových stavů, které ale mohou být ve vytvořeném projektu ignorovány pro nemožnost do nich zasahovat. Pokud nebude kontrola systému správná je možné takovýto přechod ignorovat, protože se hlavně vztahuje k testu hardwaru a vynucení této chyby by si vyžádalo zásah do testovaného zařízení. Takovýto test je výhodnější provádět až na konec aby se zamezilo zničení nebo špatné funkčnosti celého zařízení. Také v případě že je v těchto stavech detekovaný plamen, zařízení zůstává ve stavu *Idle*. Pro testování takovéhoto přechodu by se pouze doplnila smyčka vedoucí do samého stavu, ze kterého vychází.

Dalším možným stavem je stav *Trial for Ignition*, dále jen TFI, který je již možné detekovat na základě výstupů ze zařízení. TFI stav je detekovaný tím, že je otevřený pilotní ventil a zapalovací zařízení (v našem případě Hot Surface Ignitor, dále jen HSI) je aktivní. Z diagramu je patrné, že pokud v tomto stavu bude detekován plamen, zařízení přejde do dalšího stavu v hlavní smyčce. Pokud ale plamen nebude detekován, zařízení přejde do pětiminutového zpoždění a následně zkouší znovu zapálit plamen. Druhý případ je nežádoucí a proto je zde takto ošetřen, protože v TFI stavu je otevřený pilotní ventil, plyn proudí k hořáku a zařízení HSI se pokouší zapálit plamen. Po skončení pětiminutového zpoždění je celý pokus opakován.



Obr. 3.7: Stavový diagram z instalačních instrukcí[9]

Následující stav lze nazvat jako stav *Running*, ve smyslu že ohřev probíhá a plamen je zapálený. Tento stav je potvrzený otevřením hlavního ventilu, aby se dosáhlo potřebného výkonu nutného k ohřevu média. Zařízení je v tomto stavu tak dlouho, dokud není dosaženo potřebné teploty definované termostatem. Pokud je tato teplota dosažena, požadavek na topení je odebrán a tímto je zařízení odpojeno od napájení. Následně bylo možné test ukončit, ale je zde ještě přítomna chybová větev, která ošetřuje případ, kdy plamen v tomto stavu zmizí. Následuje odepnutí hlavního a pilotního ventilu a zařízení je v třívteřinovém zpoždění, dokud se nepokusí o další

zapálení plamene. Takovýto stav lze nazvat jako *Interpurge* a je definovaný tím, že zde není aktivní ani jeden výstup(přesněji ani otevřený pilotní a hlavní ventil a neaktivní HSI). Po skončení tohoto zpoždění zařízení přejde přímo do stavu TFI a pokusí se znovu o zapálení plamene.

Poslední věcí, kterou je nutné podchytit je poznámka ve stavovém diagramu, že pokud zařízení ve stavu *Running* ztratí plamen více než pětkrát za sebou, přejde do pětiminutového čekání. Toto čekání lze nazvat jako *Soft Lockout*, což znamená čekání, které není nekonečné, protože nenastala chyba, která by znemožnila funkčnost celé jednotky(takový stav by se mohl nazvat jako *Hard Lockout*).

Z analýzy stavového diagramu vyplývá, že máme k dispozici pět stavů, které jsou schopné popsat základní funkčnost zařízení a známe přechody mezi nimi, alespoň v hrubém návrhu. Jak ale bylo řečeno na konci předchozího odstavce, zařízení obsahuje také cyklický test. V kapitole 2.6.1 bylo řečeno že testovací aplikace neumožňuje přidávat podmínky pro cyklování a testování čítačů. Proto budou pro testování zařízení Smart Valve SV9501 vytvořeny dva projekty. První otestuje základní funkčnost a druhý otestuje zda opravdu zařízení po pěti ztrátách plamene ve stavu *Running* opravdu přejde do stavu *Soft Lockout*.

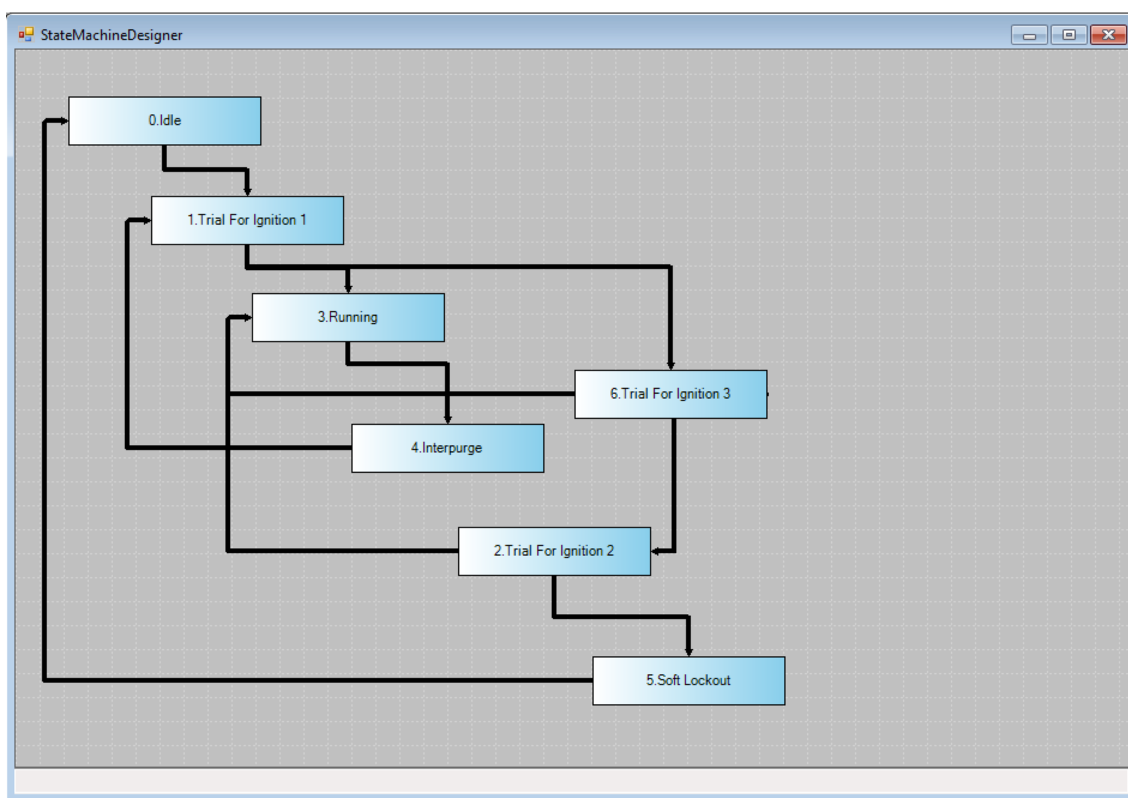
Nevyužitý výstup na modul EFT je spuštěný ve stavu *Running*, ale protože tento stav je definovaný otevření hlavního ventilu, není nutné je v celém projektu využít, pouze by potvrzoval již známý fakt.

První vytvořený projekt je na obr.3.8. Jedná se o ověření funkčnosti zařízení bez cyklického testu. Bylo vycházeno z analýzy uvedené výše. Je zde pouze jeden rozdíl a to, že samotný stav TFI je rozdělený do tří částí. Je to způsobeno samotným řízením HSI modulu, kdy v průběhu celého stavu je modul jednou vypnut, ale pilotní ventil je stále otevřený. Tento fakt byl objeven až při samotném testování a díky tomuto bylo potřebné celý projekt modifikovat. Jsou zde přítomny všechny možné přechody, které je nutné otestovat.

Hlavní smyčka je v tomto projektu relativně specifická, protože se nevrací zpět do výchozího stavu *Idle*. Bylo by možné dokončit hlavní smyčku takto, ale odpojení napájení by přineslo pouze úplné vypnutí celého zařízení a poté by muselo být znovu připojeno, aby bylo možné dále testovat a faktem také je, že vypnutá jednotka nebude žádným způsobem reagovat na impulzy od I/O karty a přidávat do takovéhoto stavu další přechody kromě přechodu který připojí napájení je relativně neefektivní. Z tohoto důvodu je hlavní smyčka ukončena stavem *Interpurge*, ze kterého se přejde znovu do stavu TFI. Aby se řidící jednotka zařízení dostala do výchozího stavu, je v tomto případě nutné přejít přes stav *Soft Lockout*, ale testovací procedura by měla být schopná otestovat i takovýto stavový diagram.

Celý projekt obsahuje jak události které se vztahují k aktivování výstupu, tak události vztahující se k vypršení časového intervalu. V první fázi se jedná o udá-

losti kdy je připojeno napájení, aktivní plamen a neaktivní plamen. Aby se zařízení dostalo do stavu *Soft Lockout*, je nutné ve stavu TFI dosáhnout vypršení časového intervalu. V této fázi je zde použita událost která dosáhne vypršení tohoto časového intervalu. Protože ale samotný stav TFI je rozdělen na tři části, časový interval je také rozdělen na tři části. Z každé třetiny stavu TFI lze opětovně přejít do stavu *Runinng*. Zde již dochází k odbočkám a dalšímu větvení, což je příhodné pro otestovaní funkčnosti testovací procedury. Další časový interval je použitý pro stav *Interpurge*, kdy má dojít po třech vteřinách k otevření pilotního ventilu a aktivování modulu HSI a tedy k přechodu do stavu TFI. Poslední časový interval je pětiminutový a znamená vypršení časového intervalu pro stav *Soft Lockout*. Z tohoto stavu nesmí vést jiný přechod než ten který vede do stavu *Idle*, protože tento stav je bezpečností a nesmí být možné jej jakkoliv přerušit, kromě odpojení napájení.

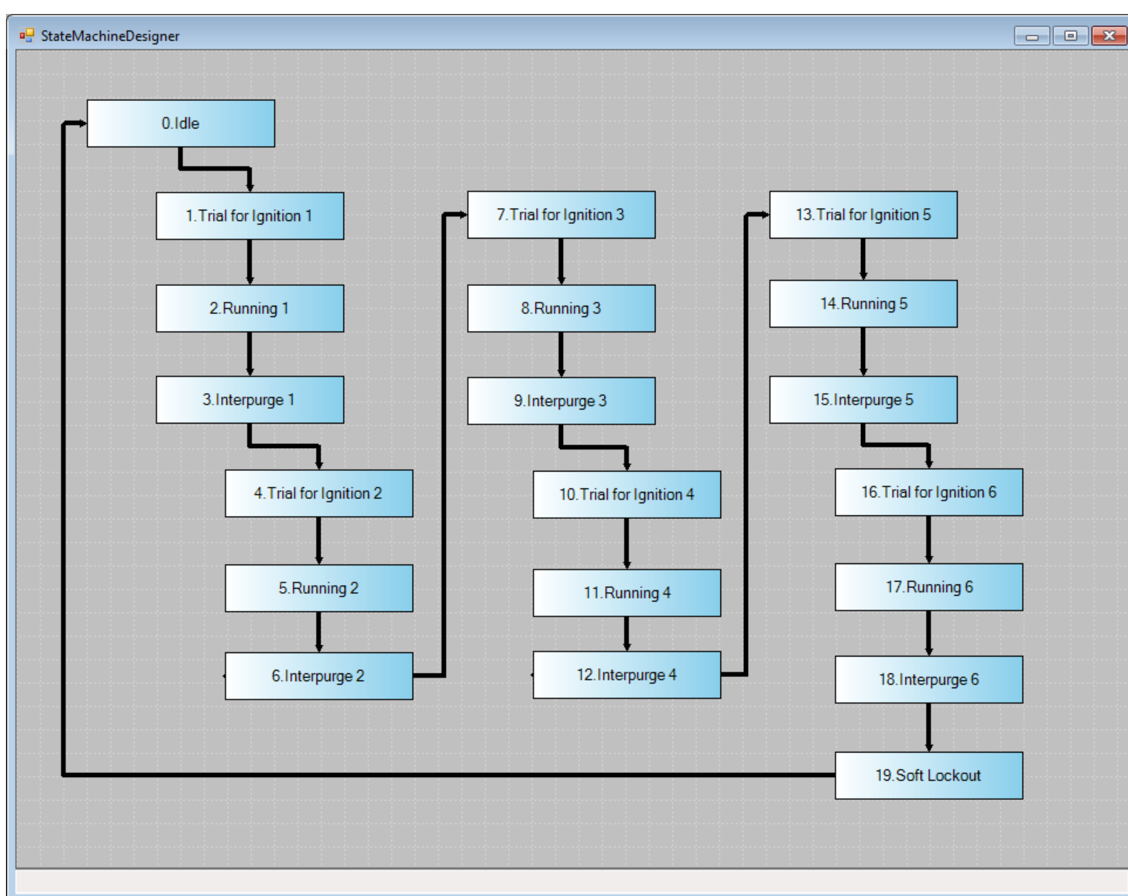


Obr. 3.8: Diagram řidícího automatu zařízení SV9501

V celém navrženém projektu je nutné počítat s faktem, že některé stavy nebude možné přesně definovat. V případě přechodu ze stavu *Soft Lockout* do stavu *Idle* lze říci, že tyto dva stavy jsou identické. Jediným způsobem jak ověřit že se jednotka nachází ve stavu *Soft Lockout* je to, že během pěti minut nesmí být aktivní žádný jiný výstup zařízení. Je známo, že pokud je zařízení ve stavu *Idle* a má aktivní požadavek na topení, otevře pilotní ventil a začne pokus o zapálení plamene. Pokud

k tomu dojde v časovém intervalu pěti minut, je stoprocentně jisté, že zařízení v tuto chvíli nebylo ve stavu *Soft Lockout*. Na těchto známých faktech jsou postavené všechny neurčité stavy a jejich následná analýza zda byly nebo nebyly provedené korektně.

Druhý vytvořený projekt je na obr.3.9. V zásadě se jedná o cyklický test, který je rozvedený do jednotlivých úseků bez jakýchkoliv odboček. Pokud je projekt blíže studován, jedná se pouze o část původního projektu, tedy o hlavní větev stavového diagramu z obr.3.8. Všechny stavy které jsou ve svém důsledku stejné, ale dle počtu provedených smyček se nacházejí dále ve stavovém diagramu, jsou pojmenovány stejným názvem, ale očíslovány tak aby bylo možné identifikovat cyklus.



Obr. 3.9: Diagram řídícího automatu zařízení SV9501 pro cyklický test

Protože výsledný diagram navržený v testovací aplikaci by byl příliš rozsáhlý, kdyby se postupovalo dle schodovitého rozmístění, je tento diagram upraven tak, aby byl co nejpřehlednější a také zabíral co nejméně místa a nebylo nutné neustále se přesouvat v grafickém prostředí aplikace. Jeden cyklus je vždy sdružen do jednoho zarovnaného bloku. Účelem testu je ověřit zda opravdu po tom, co je zařízení odebrán plamen ve stavu *Running* a tento případ nastal již po šesté, zařízení přejde do

stavu *Soft Lockout*. Stejně jako v předchozím případě je tento test potvrzen tím, že se během pěti minut kdy je předpokládáno že zařízení je ve stavu *Soft Lockout*, nereaguje jednotka žádným způsobem na jiné podněty. Kdyby nastal případ, že hodnota časovače bude vyšší než předpokládaná, test nebude správně dokončen, protože při stavu *Soft Lockout* dojde k aktivování pilotního ventilu a modulu HSI, což se v tomto stavu na základě definice nemůže stát a na druhou stranu, pokud hodnota časovače bude nižší než předpokládaná, zařízení ve stavu *Soft Lockout* bude a nebude reagovat žádnými podněty a tím pádem dojde také k havárii testu. Ve všech případech je tedy možné zachytit chybu jednotky, pokud je vytvořený projekt v souladu s definovanou funkčností zařízení.

Následné otestování obou projektů by mělo přinést výsledky, zda je zařízení funkční. Protože ale jsou projekty vytvořené přesně v souladu s definovanou funkčností jednotky a u testované jednotky je stoprocentní jistota že tato jednotka je funkční, chyby které mohou při testování nastat jsou způsobené navrženou testovací procedurou. Všechny tyto testy uvedené v celé kapitole se zde hlavně zaměřují na ověření funkčnosti testovací procedury, ne na ověření funkčnosti jednotky Smart Valve SV9501.

3.2.2 Výsledky testů

Po provedení testů na zařízení SV9501 je nutné vyvodit ze záznamů určité závěry. Výsledky prvního funkčního testu jsou na obr.3.10. Ze zobrazeného textového souboru je vidět, že procedura provedla dva testovací cykly. V tomto případě je tato informace mírně nepotřebná, jelikož z navrženého diagramu pro tento test bylo předpokládáno, že do výchozího stavu se jednotka dostane pouze dvakrát, což bylo nakonec potvrzeno ve výsledku. Z obecné informace na konci testu lze říci že test byl proveden kompletní a všechny přechody byly otestovány. Z toho vyplývá, že testovací procedura je schopná testovat i takovéto stavové automaty s více odbočkami.

Aby bylo možné dále analyzovat test, je nutné se podívat podrobněji na testované přechody. V případě přechodu z *Interpurge* do TFI je předpokládaná doba přechodu tři vteřiny, s ohledem na definovaný stavový diagram z instalačních instrukcí. Je nutné zde ale počítat se zpožděním, které je patrné z tohoto diagramu, protože fakticky zařízení přechází z *Interpurge* do stavu kde se interně testuje celé zařízení a tento stav trvá tři vteřiny. V konečném důsledku tedy tento přechod trval šest vteřin, nicméně je nutné zde brát v potaz, že v tomto přechodu je také obsažen test samotného zařízení a celkový čas je nutné rozdělit mezi tyto dvě části. Způsob čtení výsledků je třeba interpretovat na základě skutečného stavového diagramu a mít v patrnosti strukturu navrženého testu. Pro podrobnější popis automatu by bylo nutné znát podrobnější informace o zařízení, ale tyto informace nejsou dostupné pro

širokou veřejnost a jsou majetkem společnosti Honeywell.

Další podrobností na kterou se lze zaměřit je samotný přechod do stavu *Soft Lockout* na konci prvního testovacího cyklu. Trvání celého cyklu TFI lze spočítat sečtením jednotlivých naměřených intervalů, což činí přibližně devadesát vteřin a tento čas je tímto potvrzený. Ve druhém cyklu lze vidět, že testovací procedura testuje přechody z jednotlivých části stavu TFI do stavu *Running*. Dobu trvání stavu *Soft Lockout* je možné následně vypočítat sečtením doby trvání přechodu z tohoto stavu do výchozího stavu(*Idle*) a následně k němu připočíst dobu trvání přechodu z výchozího stavu do následujícího stavu. Pro stav *Soft Lockout* byl totiž nastaven nižší čas vypršení časovače než je skutečný, aby bylo možné v následujícím stavu změřit přesnou dobu trvání tohoto stavu. Od výsledku je také nutné odečíst tři vteřiny trvání stavu kdy je kontrolována samotná jednotka, jak již bylo řečené výše. Výsledný čas se velice přibližuje ke zmíněným pěti minutám, což je dle instalačních instrukcí přesná doba trvání tohoto stavu. Je možné sledovat postup postupného učení testovací procedury a rozhodování o výběru další cesty. Zařízení často prochází stavy *Running*, *Interpurge* a TFI, ale i přesto je testovací procedura schopná vybrat postup tak, aby se dostala k ještě neprovedeným událostem.

```
Test started 5/4/2015 6:59:51 AM
Device: SV9501
User: Stanislav Horky

Legend: From State -> Next State (Event that caused transition) : Duration of transition

Test run 0
06:59:57 Idle->Trial For Ignition 1(Power On): 6384
07:00:00 Trial For Ignition 1->Running(Flame ON): 2513
07:00:04 Running->Interpurge(Flame Off): 4169
07:00:11 Interpurge->Trial For Ignition 1(Timer Expires 1): 6676
07:00:41 Trial For Ignition 1->Trial For Ignition 3(Timer Expires 2): 29864
07:01:06 Trial For Ignition 3->Trial For Ignition 2(Timer Expires 2): 25037
07:01:40 Trial For Ignition 2->Soft Lockout(Timer Expires 2): 34149
07:06:20 Soft Lockout->Idle(Timer Expires 3): 280000

Test run 1
07:06:45 Idle->Trial For Ignition 1(Power On): 25415
07:07:15 Trial For Ignition 1->Trial For Ignition 3(Timer Expires 2): 29860
07:07:18 Trial For Ignition 3->Running(Flame ON): 2256
07:07:22 Running->Interpurge(Flame Off): 4128
07:07:29 Interpurge->Trial For Ignition 1(Timer Expires 1): 7117
07:07:59 Trial For Ignition 1->Trial For Ignition 3(Timer Expires 2): 29767
07:08:24 Trial For Ignition 3->Trial For Ignition 2(Timer Expires 2): 24947
07:08:26 Trial For Ignition 2->Running(Flame ON): 2248

Passed
```

Obr. 3.10: Výsledky prvního testu stavového automatu zařízení SV9501

Výsledky cyklického testu jsou na obr.3.11. Podle obecného výsledku lze říct, že pokud jednotka SV9501 ve stavu *Running* šestkrát ztratí přítomnost plamene(respektive

plamen není možné detekovat), přejde do stavu *Soft Lockout*. Celý test proběhl v jednom jediném testovacím cyklu, protože v navrženém diagramu nejsou žádné odbočky. Na postupu testu je z výsledků patrné, že časy jednotlivých přechodů se liší velmi málo, ale je zde možné vidět že jsou tyto časy podobné a referují tak na stejný přechod.

```
Test started 5/4/2015 9:46:58 AM
Device: SV9501_Cycle
User: Stanislav Horky

Legend: From State -> Next State (Event that caused transition) : Duration of transition

Test run 0
09:47:04 Idle->Trial for Ignition 1(Power On): 6484
09:47:07 Trial for Ignition 1->Running 1(Flame On): 2505
09:47:11 Running 1->Interpurge 1(Flame Off): 4105
09:47:18 Interpurge 1->Trial for Ignition 2(Timer Expires 1): 6710
09:47:20 Trial for Ignition 2->Running 2(Flame On): 2514
09:47:24 Running 2->Interpurge 2(Flame Off): 4087
09:47:31 Interpurge 2->Trial for Ignition 3(Timer Expires 1): 6728
09:47:34 Trial for Ignition 3->Running 3(Flame On): 2496
09:47:38 Running 3->Interpurge 3(Flame Off): 4107
09:47:45 Interpurge 3->Trial for Ignition 4(Timer Expires 1): 6774
09:47:47 Trial for Ignition 4->Running 4(Flame On): 2494
09:47:51 Running 4->Interpurge 4(Flame Off): 4082
09:47:58 Interpurge 4->Trial for Ignition 5(Timer Expires 1): 6727
09:48:01 Trial for Ignition 5->Running 5(Flame On): 2519
09:48:05 Running 5->Interpurge 5(Flame Off): 4056
09:48:12 Interpurge 5->Trial for Ignition 6(Timer Expires 1): 6744
09:48:14 Trial for Ignition 6->Running 6(Flame On): 2505
09:48:18 Running 6->Interpurge 6(Flame Off): 4074
09:48:21 Interpurge 6->Soft Lockout(Timer Expires 1): 3000
09:53:01 Soft Lockout->Idle(Timer Expires 3): 280000

Passed
```

Obr. 3.11: Výsledky druhého testu stavového automatu zařízení SV9501

V konečném důsledku lze tedy říci, že testovací procedurou navrženou v této práci lze otestovat více zařízení a je možné zde nakonfigurovat i cyklické testy. Vše záleží na tom, zda uživatel dokáže aplikaci využít správně. V této kapitole bylo otestováno reálné zařízení Smart Valve SV9501 a pomocí dvou testů byla prokázána jeho funkčnost a navíc byly změřené časy jednotlivých přechodů, z čehož lze vycházet při analýze přesnosti časování.

4 ZÁVĚR

První část práce obsahuje teoretický rozbor a definice pojmů. Jsou zde rozebrány řidící jednotky realizované stavovými automaty a způsoby jejich popisu a nastíněna struktura tabulky, kterou bude možné použít jako jednu ze vstupních informací pro aplikaci. Dále je zde vybrán jazyk XML jako formát výstupních dat. Poslední částí teoretického úvodu je náhled na strukturu systému, který bude provádět autonomní testování.

Ve druhé části je popsána samotná aplikace. Nejprve je nastíněn koncept celé aplikace, tedy že vstupní informací bude tabulka vytvořená v aplikaci MS Excel, nebo bude uživateli umožněno vytvářet diagramy řidícího stavového automatu zcela od začátku, bez použití vstupní tabulky. Proces vytváření XML struktury je zde ukázán vývojovým diagramem. Aplikace používá projekt Diagram.NET, který je volně dostupný a slouží k vykreslování objektů, tvoření diagramů a interakci s těmito objekty. Vytvořený grafický editor umožňuje přidávat, odstraňovat a modifikovat stavy a přechody(dále objekty), přidávat události a proměnné, tedy vše co je potřeba aby uživatel mohl navrhnout řidící stavový automat dle svých představ a požadavků. Celý editor byl koncipován tak, aby jeho používání a ovládání bylo co nejsrozumitelnější, je možné označovat objekty a zjišťovat tak jejich napojení na ostatní složky automatu a také je možné vypisovat detailní informace o objektech, které reprezentují stavy automatu. Dále je zde rozveden vývoj a struktura testovací procedury, která bude testovat fyzická zařízení. Zde je nastíněn základní princip celého testování, konfigurace testu, zapisování výsledků a hlavně je zde popsán postup, jakým testovací procedura prochází samotný stavový diagram a na základě jakých parametrů se rozhoduje v průběhu testu dále, aby bylo možné provést kompletní test. Tyto postupy jsou v této práci naznačené ve vývojovém diagramu, který se zabývá samotnou testovací procedurou a její kompletní funkčností. Na konci této části jsou nastíněné další možné modifikace celé aplikace, což znamená směry kterými by se měl další vývoj takovéto aplikace ubírat.

Poslední kapitola obsahuje praktický příklad. Jsou zde otestovány dvě řidící jednotky. Pro každou jednotku jsou provedené dva testy. V prvním případě je testováno zařízení STK500 s procesorem ATmega16 a je zde prokázána funkčnost navržené testovací procedury. První test této jednotky je zaměřen na samotnou funkčnost a další test potvrzuje to, že testovací procedura dokáže zaznamenat chybový stav a následně v záznamu informovat uživatele tak, aby byl schopný odhalit jak tento stav nastal a aby byl na základě této informace schopný tuto chybu odhalit a eliminovat a aby dokázal určit zda tato chyba byla vyvolána samotným testovaným zařízením, nebo vytvořeným stavovým diagramem a jeho nakonfigurováním. Druhou testovanou jednotkou je řidící jednotka Smart Valve SV9501 vytvořená ve vývojovém centru firmy

Honeywell. V prvním testu této jednotky byla ověřena její funkčnost, stejně tak jako funkčnost testovací procedury a to, že je schopná testovat i reálná zařízení, která se vyrábí a prodávají. V druhém testu byla potvrzena možnost, že pomocí testovací procedury lze vytvořit a otestovat cyklický test, aniž by bylo nutné vytvářet za tímto účelem novou testovací proceduru.

Vytvořená aplikace tedy umožňuje grafický návrh stavových diagramů dle uživatelových nároků a je možné s její pomocí vytvořit škálu testovacích projektů, které se mohou vztahovat k různým zařízením. Aplikace je schopná na základě vytvořených postupů testovat tato zařízení. Pomocí konfigurace je možné komunikovat s I/O kartou a tím komunikovat i s fyzickým zařízením. Dále prezentuje grafický postup samotného testu a na samotném konci vygeneruje přehledný záznam o průběhu celého testu, který je strukturovaný tak aby byl přehledný a umožnil uživateli další analýzu.

Přesnost časování je závislá na vlastnostech použitých I/O karet. Záleží tedy na výběru elementů v systému, který je naznačen v teoretickém úvodu. Tato práce je studií funkčnosti navrženého systému a vytvořená aplikace je pouze interpret akcí a reakcí, ale nedokáže ovlivnit kvalitu celého systému. Výsledky testů potvrzují, že tato funkčnost byla prokázána a je tedy možné na tomto systému vytvořit podobné autonomní testovací stanoviště. Dále také byl potvrzena funkčnost konceptu popsaného na začátku teoretického úvodu této práce.

LITERATURA

[1] BRAY, T., PAOLI, J., SPERBERG-MCQUEEN, J.C. *Extensible Markup Language (XML) 1.0* [online]., 10.2.1998, [cit. 26. 11. 2014]. Dostupné z URL: <http://www.w3.org/TR/1998/REC-xml-19980210.pdf>.

[2] SOBRINHO, D.L. *Diagram.Net* [online]., 2004, [cit. 7. 12. 2014]. Dostupné z URL: <http://www.dalssoft.com/diagram/default.aspx>.

[3] Windows Forms *Microsoft Developer Network* [online]., 2014, [cit. 7. 12. 2014]. Dostupné z URL: <http://msdn.microsoft.com/en-us/library/dd30h2yb%28v=vs.110%29.aspx>.

[4] Multiple-Document Interface (MDI) Applications *Microsoft Developer Network* [online]., 2014, [cit. 7. 12. 2014]. Dostupné z URL: <http://msdn.microsoft.com/en-us/library/xyhh2e7e(v=vs.110).aspx>.

[5] Embedded system *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-, 15. 11. 2014 [cit. 20. 11. 2014]. Dostupné z URL: <http://en.wikipedia.org/wiki/Embedded_system>.

[6] Tree (data structure) *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-, 24. 11. 2014 [cit. 9. 12. 2014]. Dostupné z URL: <http://en.wikipedia.org/wiki/Tree_(data_structure)>.

[7] Multitier architecture *Wikipedia: the free encyclopedia)* [online]. San Francisco (CA): Wikimedia Foundation, 2001-, 19. 11. 2014 [cit. 9. 12. 2014]. Dostupné z URL: <https://en.wikipedia.org/wiki/Multitier_architecture>.

[8] State diagram *Wikipedia: the free encyclopedia)* [online]. San Francisco (CA): Wikimedia Foundation, 2001-, 5. 12. 2014 [cit. 9. 12. 2014]. Dostupné z URL: <http://en.wikipedia.org/wiki/State_diagram>.

[9] SV9501/SV9502/SV9601/SV9602 SmartValveTM System Control *Honeywell* [online]., 10.2.1998, [cit. 26. 4. 2015]. Dostupné z URL: <https://www.forwardthinking.honeywell.com/related_links/combustion/universal_smart_valve/install/69_1270.pdf>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

MDI	Rozhraní pro více dokumentů – Multiple-Document Interface
XML	Rozsáhlý značkovací jazyk – Extensive Markup Language
SGML	Standartní generalizovaný značkovací jazyk – Standard Generalized Markup Language
I/O	Vstupně výstupní moduly – Input output modules
TFI	Pokus o zapálení plamene – Trial for Ignition
HSI	Zapalovací zařízení na principu žhavého povrchu – Hot Surface Ignitor
EFT	Jednotka ovládající ventilátory – Electronic Fan Timer
TTL	Tranzistorově-tranzistorová logika – Transistor-transistor-logic

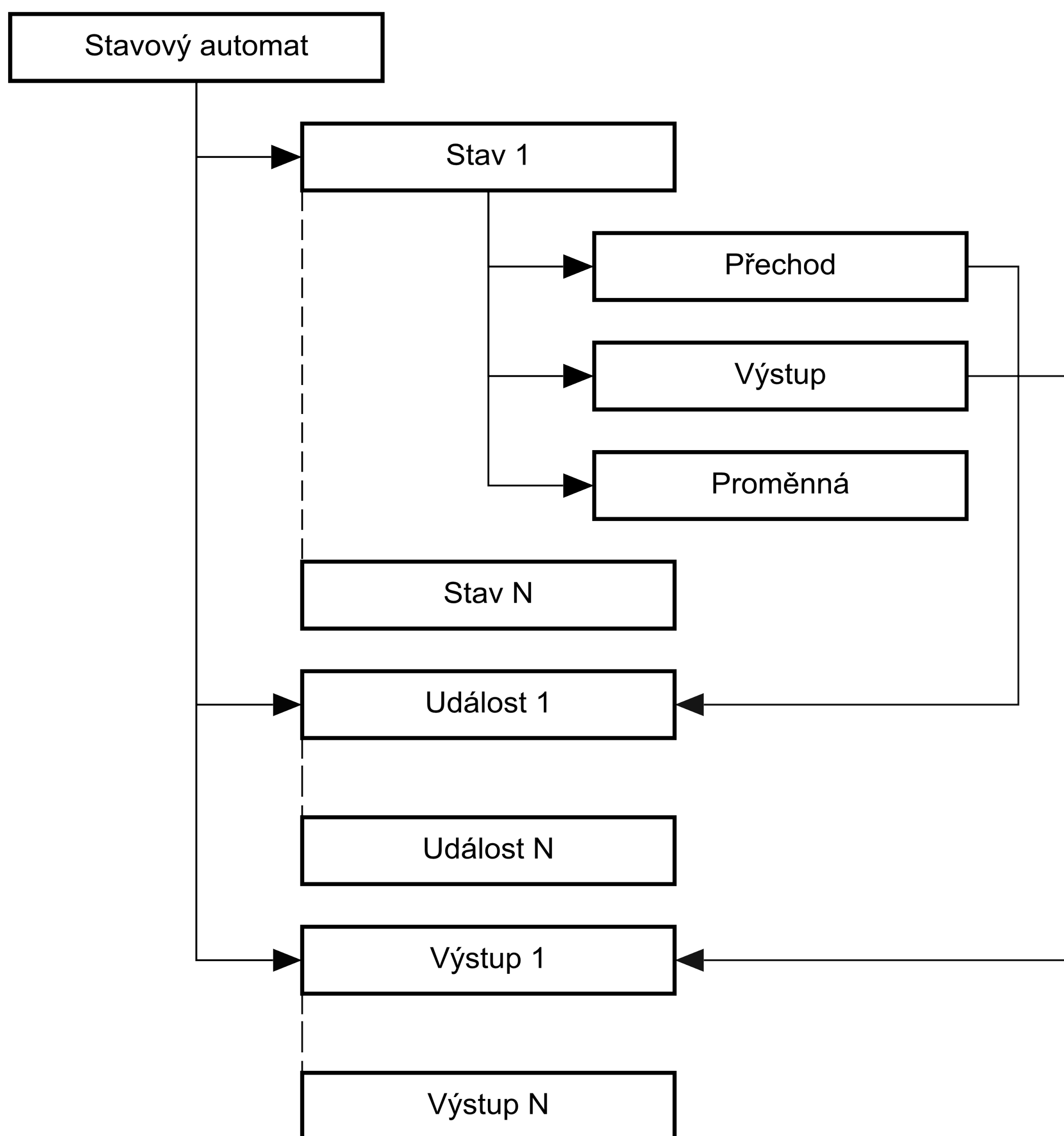
SEZNAM PŘÍLOH

A Stavový automat **74**

A STAVOVÝ AUTOMAT

SH1001													
			Events					Outputs			Variables	
number	symbol	state	Internal Error	Relay1 On	Relay2 On	ADC value > Threshold1	Timer Expires	Relay1	Relay2	A/D	Timer	Threshold1
			IER	R1O	R2O	ADC	TE	RL1	RL2	ADC	TM	TH1
0	IDL	Idle	SLO		CHW	IDL	IDL		1	1	1	0,2
1	CHW	Check Hardware	SLO			IDL	ST1		1	1	12	0,2
2	ST1	State 1	SLO			IDL	CHR		1	1	5,2	1,5
3	CHR	Check Run	SLO	RUN		SLO	SLO	1		1	1,8	0,4
4	RUN	Run	SLO		CHW	RUN	ST2		1	1	1,5	0,4
5	ST2	State 2	SLO				ST3		1		10	1,2
6	ST3	State 3	SLO	ST5			ST4	1			5	0,4
7	ST4	State 4	SLO		ST2		ST5		1		6	0,4
8	ST5	State 5	SLO				SLO				1	0,5
9	SLO	Soft Lockout					/Reset				500	

Obr. A.1: Stavová tabulka automatu



Obr. A.2: Postup vytváření struktury v jádře