TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: X2612 – Elektrotechnika a informatika
Studijní obor: 1234T567 – Automatizované řízení

Editor a simulátor stavových diagramů

Editor and simulator of state diagrams

Diplomová práce

Autor: **Miroslav Hajfler**
Vedoucí práce: RNDr. Pavel Satrapa, Ph.D.

V Liberci 6. 5. 2007

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Katedra aplikované informatiky Akademický rok: 2006/2007

ZADÁNÍ DIPLOMOVÉ PRÁCE

Jméno a příjmení: Miroslav Hajfler

studijní program: M 2612 – Elektrotechnika a informatika

obor: 3902T005 – Automatické řízení a inženýrská informatika

Vedoucí katedry Vám ve smyslu zákona o vysokých školách č.111/1998 Sb. určuje tuto diplomovou práci:

Název tématu: **Editor a simulátor stavových diagramů**

Zásady pro vypracování:

1. Seznamte se s problematikou stavových diagramů a jejich aplikací, především při návrhu komunikačních protokolů.
2. Navrhněte datové formáty pro ukládání stavových diagramů a časových sledů událostí
3. Vytvořte grafickou aplikaci, umožňující interaktivní vytváření a editaci stavových diagramů a následnou simulaci jejich činnosti.
4. Ověřte funkčnost aplikace na konkrétních stavových diagramech.

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé diplomové práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom(a) toho, že užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Diplomovou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum

Podpis

Poděkování

Chtěl bych touto cestou poděkovat všem, kdo se nějak podíleli na této práci. Zvláště pak své rodině za poskytnutou podporu a zázemí a svému vedoucímu diplomové práce za poskytnuté rady a připomínky k práci.

Abstrakt

Diplomová práce má za cíl vytvořit programové prostředí, ve kterém lze snadno vytvářet a upravovat stavové digramy a následně simulovat jejich činnost. Využití programu by mělo být především při simulování funkce komunikačních protokolů, ale je použitelný i pro obecné stavové automaty (např. pro zjišťování akceptovatelnosti daného slova) nebo Mooreho automaty.

Diplomová práce nejprve seznamuje s teorií stavových digramů. Dále se zabývá využitím stavových digramů při návrhu komunikačních protokolů. V druhé části se DP zaměřuje na vlastní grafickou aplikaci. Vysvětluje základní funkce programu, důležité algoritmy a datové formáty použité pro ukládání stavových digramů a posloupností událostí. V poslední části DP hodnotí výsledky použití programu na vytváření a simulování konkrétních stavových digramů.

Abstract

Diploma Thesis has an aim create software interface, allows easy creating and editing state diagrams and next simulating its function. Program exploitation should been especially for simulations of functions of communication protocols, but its useful for general state automats (for example for recognition ability of accept query word) or Moore's automats too.

Diploma Thesis at first makes familiar theory of state diagrams. Next it deals with state diagrams exploitation for design of communication protocols. In second part DT focus on incident graphic application. It explicates basic functions of program, important algorithms and data format used for save state diagrams end events sequence. In the last part DT appreciates results of using program for creation and simulation concrete state diagrams.

Obsah:

Úvod

Hlavním úkolem této diplomové práce, bylo vytvořit prostředek pro snadnou tvorbu, úpravu a následnou simulaci stavových digramů. Pro programování tohoto prostředku jsem zvolil programovací prostředí Borland Delphi 2005. Tomuto prostředí jsem dal přednost, protože ze všech možných variant, s ním mám nejvíce zkušeností a také se v něm snadno vytváří nové třídy, což jsem využil například pro vytvoření třídy TState1, která je důležitou součástí mého programu. Program by měl být použitelný pro simulaci komunikačních protokolů, Mooreho automatů nebo stavových automatů.

1 Teorie stavových diagramů

1.1 Stavový diagram (state transition diagram)

Stavový diagram patří mezi klasické a osvědčené nástroje objektového modelování používané pro znázorňování chování systému. V každém systému dochází s ubíhajícím časem k novým událostem a systém se díky nim mění. V průběhu interakce s uživateli nebo jinými systémy procházejí jednotlivé objekty, z nichž je systém sestaven, nezbytnými změnami. Pro modelování systémů je nutné mít k dispozici prostředek pro modelování těchto změn. Jednou z možností, jak popsat systémové změny, je prohlásit, že jeho jednotlivé objekty mění v čase svůj stav. Tato změna stavu je reakcí na událost či prostý fakt, že čas ubíhá.

Stav objektu lze charakterizovat tak, že se jedná o konkrétní situaci, která nastala za doby života objektu. Tato situace je dále charakterizována tak, že během ní objekt vyhovuje nějaké podmínce, realizuje konkrétní operaci nebo třeba jen čeká na příchod události. Každý stav ve stavovém diagramu odpovídá jedné možné hodnotě stavového atributu nebo (v některých případech) rozsahu hodnot daného atributu. Jako stavový atribut je označována vlastnost (nebo souhrn vlastností) daného objektu, jejíž hodnota ovlivňuje stav objektu. Přechod mezi stavy je pak vlastně vyjádřením změny hodnot tohoto atributu. Přechod mezi jednotlivými stavy bývá vyvolán událostí, jež je většinou podnětem z vnějšího okolí, nejčastěji ve formě zprávy zaslané příslušnému objektu nebo změnou vnějších proměnných.

Diagramu stavů a přechodů, jak je někdy tento prostředek nazýván, je ideální pro modelování objektu se stavovým atributem s následujícími dvěma charakteristikami: atribut může nabývat pouze málo hodnot a má omezení týkající se povolených přechodů mezi zmíněnými hodnotami. Objekty nesplňující tyto podmínky lze samozřejmě také modelovat pomocí stavového digramu, ale tento model může být velice složitý, zmatečný a jeho použitelnost bude tudíž problematická. Definici stavů objektu je potřeba provést s ohledem na to, jaké stavy mají v daném systému význam a jaké má tedy smysl rozlišovat.

Stavový digram zobrazuje povolené stavy, které mohou objekty daného systému dosáhnout, a povolené přechody mezi nimi. Mezi stavy se kromě běžných stavů vyskytují dva speciální druhy. Počáteční stav je stav, ve kterém se objekt nachází na počátku svého životního cyklu. Koncový stav je objekt, kde končí životní cyklus

objektu, ale je možné ho využít také při zjišťování akceptovatelnosti nějakého slova nad příslušným stavovým automatem.

Stavové diagramy jsou důležité, protože pomáhají analytikům, autorům návrhu i vývojářům pochopit chování objektu obsažených v systému.

1.2 Možnosti reprezentace stavových digramů

Reprezentace stavového diagramu může být, buď ve formě tabulky přechodů nebo častěji ve formě orientovaného ohodnoceného grafu. Tabulka přechodů popisuje, které stavy jsou počáteční, které koncové a kam vedou přechody vyvolané možnými událostmi.

	0	1
←A	B	E
→B	D	A
C	E	D
D	F	E
←E	E	B
F	A	A

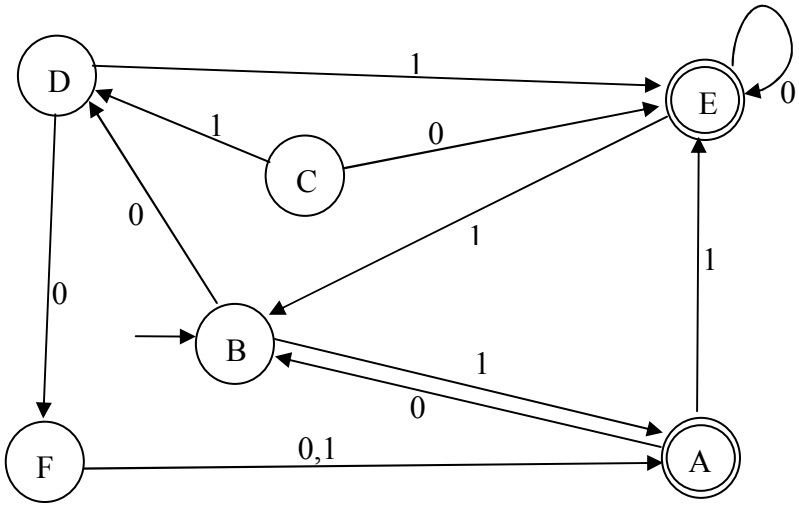
Obr. 1 - Příklad tabulky přechodů

Reprezentace orientovaným ohodnoceným grafem se používá častěji, protože usnadňuje pochopení funkce stavového diagramu. Vrcholy grafu jsou jednotlivé stavy. Hrany představují přechody a jsou označeny událostmi, které je vyvolávají. Existence více než jedné události u hrany znamená, že přechod může spustit kterákoli z uvedených událostí.

1.3 Druhy stavových digramů

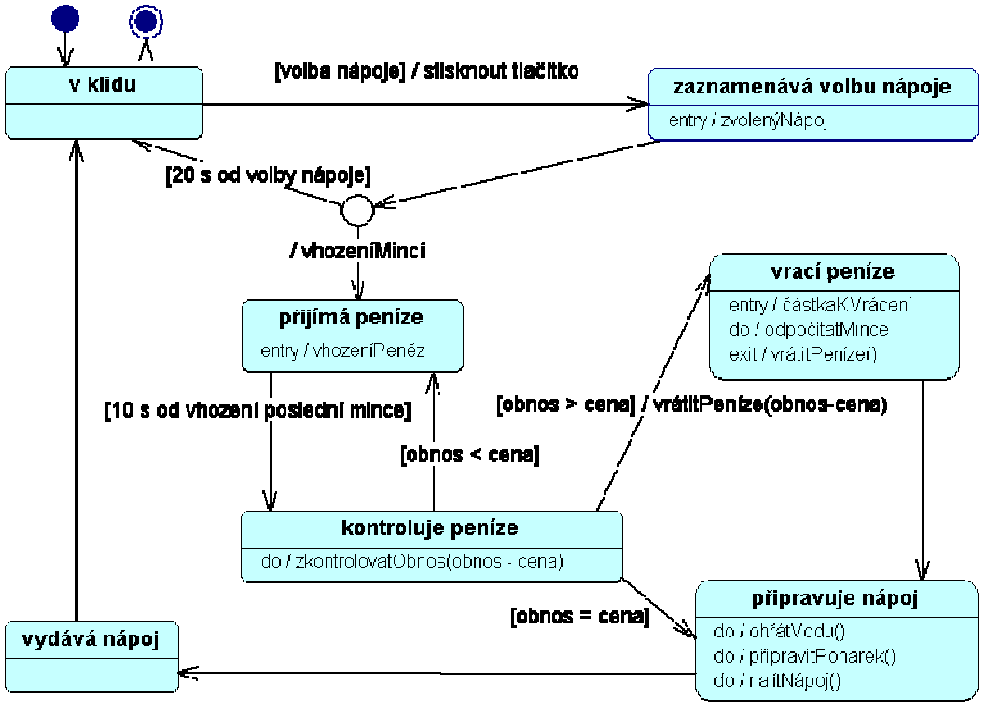
Existuje řada forem stavových digramů, které se navzájem mírně odlišují sémantikou. Často se stavový diagram zaměňuje se stavovým automatem, ale stavový diagram je jen pomůcka k znázornění automatu. V tomto případě je stav reprezentován ikonou ve formě kruhu, ve kterém je zobrazen jeho název. Přechod mezi stavy je znázorněn plnou šipkou mířící od výchozího stavu k novému. Tato šipka se také nazývá stavová trajektorie. Pokud přechod vede do stavu, ze kterého vychází, kreslí se smyčka.

Počáteční stav bývá označen šipkou směřující do něj. Koncový stav šipkou směřující z něj nebo dvojitým kroužkem.



Obr. 2 - Příklad stavového diagramu pro koncový automat vyjádřený orientovaným grafem

V jazyce UML se vychází ze stavových diagramů Davida Harela. Od diagramů popisujících konečné automaty se liší v několika aspektech. Stavy jsou zobrazeny jako obdélníky se zakulacenými rohy. Počáteční stav je vyznačen černou tečkou a v tomto stavu objekt netráví žádný čas a automaticky přechází do dalšího stavu. Koncový stav je označený černou tečkou s bílým okrajem. Stavové diagramy v UML jsou ale především rozšířené oproti diagramům koncových automatů. Symboly reprezentující stavy a přechody je možno doplňovat dalšími informacemi. Ikona stavu může být rozdělena do tří oblastí. V první oblasti je název stavu(tato oblast je jako jediná povinná), druhá oblast obsahuje stavové proměnné a třetí zobrazuje seznam činností. V UML je možno stavům přiřadit různé akce, které je třeba vykonat při vstupu do stavu nebo výstupu z něj. Tyto akce s podmínkami jsou právě v seznamu činností. Podobně lze akce přiřazovat i přechodům, v tom případě je stavová trajektorie označena událostí, která ho vyvolává (stejně jako u koncových automatů), ale za událostí oddělena lomítkem následuje požadovaná akce. Stavový diagram navíc může obsahovat vnořené nebo souběžné stavy.

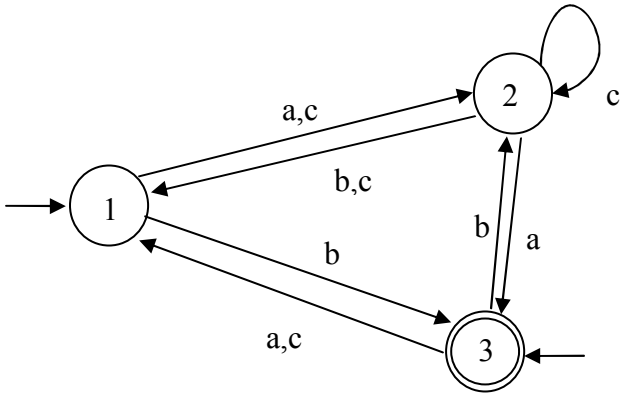


Obr. 3 - Příklad stavového diagramu v jazyce UML

Způsob kreslení stavových diagramů používaný v UML je pro můj úkol zbytečně složitý. Pro svůj program jsem zvolil jednodušší způsob kreslení stavových diagramů, vycházející z popisování stavových automatů. Liší se jen v maličkostech. Počáteční stav je barevně odlišen, koncový stav má silnější okraj a přechody, které vedou do stejného stavu, ze kterého vycházejí, nejsou zobrazeny.

1.4 Determinismus stavových digramů

Jednou ze zásad při navrhování stavových digramů je, že výsledný diagram by měl být deterministický. Deterministický stavový diagram má právě jeden počáteční (vstupní) stav a alespoň jeden koncový (výstupní) stav. Pro jednu událost nesmí směřovat přechod do více než jednoho stavu. Stav musí mít definovány přechody pro všechny přípustné události. Při přechodu nemusí dojít ke změně stavu – tzv. setrvání stavu.



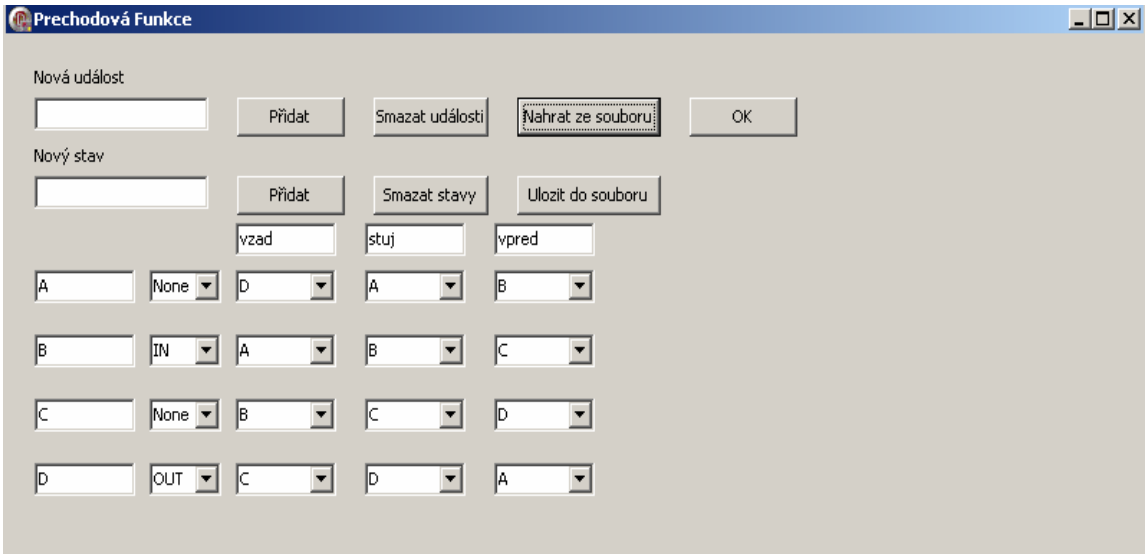
Obr. 4 - Příklad nedeterministického stavového diagramu

Nedeterminismus příkladového diagramu je způsoben dvěma počátečními stavy a tím, že ze stavu 2 vedou dva přechody vyvolané událostí c. Nedeterministické stavové diagramy se často používají u konečných automatů. Existuje pro ně způsob převodu na deterministické.

2 Ovládání aplikace

2.1 Vytváření a upravování stavových digramů

Program umožňuje vytvořit vlastní stavový diagram nebo nahrát nějaký dříve vytvořený. K vytvoření nového stavového digramu slouží položka *Přechodová funkce* v menu *Edit*. Po stisku položky se otevře formulář, ve kterém je možno upravit tabulku přechodů.



Obr. 5 - Formulář s tabulkou přechodů

V prvním sloupci jsou názvy stavů. V druhém sloupci jsou rolovací menu, pomocí nichž lze nastavovat status příslušných stavů. Status stavu může být neutrální, vstupní, výstupní nebo vstupně/výstupní. Program samozřejmě neumožňuje zvolit více než jeden vstupní stav. Pokud uživatel zvolí stav jako vstupní nebo vstupně/výstupní a jiný stav už je takto nastaven, status původního stavu se změní ze vstupního na neutrální nebo vstupně/výstupního na výstupní.

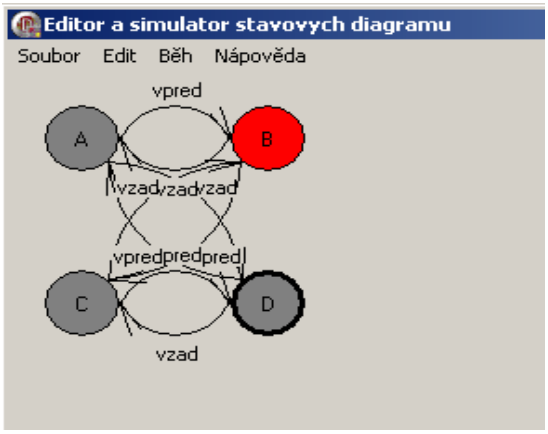
V prvním řádku jsou názvy událostí, které mohou vyvolat přechod u zadaných stavů. Ve zbývajících řádcích a sloupcích jsou rolovací menu, které umožňují nastavit stav, do kterého má směřovat přechod ze stavu na příslušném řádku, vyvolaný událostí na příslušném sloupci. Implicitně jsou nastaveny všechny přechody do stejného stavu, ze kterého vycházejí. Jsou tedy nastaveny na setrvání stavu. Je to kvůli tomu, že ke spoustě událostí může dojít pouze tehdy, pokud je objekt v určitém stavu, tudíž spousta stavů bude mít aktivní přechod (vede ke změně stavu) jen pro poměrné málo událostí.

Nová událost nebo nový stav se přidává pomocí textových vstupů *Nová událost* a *Nový stav* a pomocí příslušných tlačítek *Přidat*. Stejně jako tato tlačítka funguje i odenterování ve vstupech. Při pokusu o přidání nové položky je testováno, zda je název platný, tudíž jestli není prázdný nebo jestli už neexistuje položka stejného názvu.

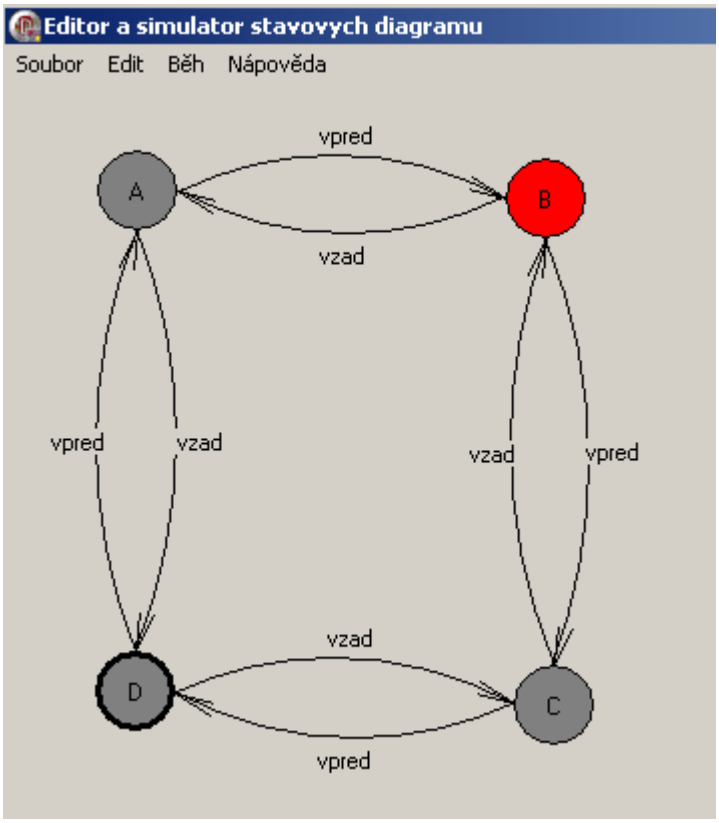
Všechny stavy nebo události lze samozřejmě hromadně smazat příslušnými tlačítky. Jednotlivé stavy nebo události lze mazat nebo měnit jejich názvy pomocí menu, které se zobrazí po stisku druhého tlačítka na příslušném názvu. Po přidání nového stavu nebo změně názvu stávajícího dojde k automatické úpravě nabídek všech rolovacích menu. Pokud se smaže stav, do kterého vede nějaký přechod, tak se tento přechod změní na setrvání stavu, protože v mém stavovém diagramu nejsou nedefinované přechody (jedná se o deterministický stavový diagram).

Dokončení tvorby tabulky přechodů se stvrdí stiskem tlačítka *OK*. Po jeho stisku se všechny potřebné informace uloží do paměti, dojde k zavření tohoto formuláře a vytvoření požadovaných stavů.

Algoritmus pro ideální rozmístění použitelný pro jakýkoli stavový diagram neexistuje. Přehledné rozmístění konkrétních stavů závisí na jejich počtu a hlavně na propojení stavů přechody. Různé stavové digramy tedy většinou musí vypadat jinak. V tomto programu jsou stavy rozmístěny podle jednoduchého algoritmu do matice n x n (poslední řádek není většinou kompletní). Toto rozmístění rozhodně není ideální a skoro jistě při něm dojde ke křížení stavových trajektorií. Stavový diagram je díky tomu dosti nepřehledný a je potřeba, aby uživatel rozmístění upravil vhodně pro daný konkrétní diagram. Přemisťování stavů je řešeno pomocí jednoduché metody Drag&Drop. Stačí stisknout levé tlačítko myši na stavu, který chceme přemístit, tím dojde k jeho uchopení. Při stále stisknutém tlačítku stav pomocí pohybu myši přesuneme na novou pozici. Po uvolnění tlačítka dojde k puštění stavu. Při pohybu stavu dochází průběžně k překreslování stavových trajektorií. Tímto způsobem může uživatel upravit stavový diagram do přehledné podoby.

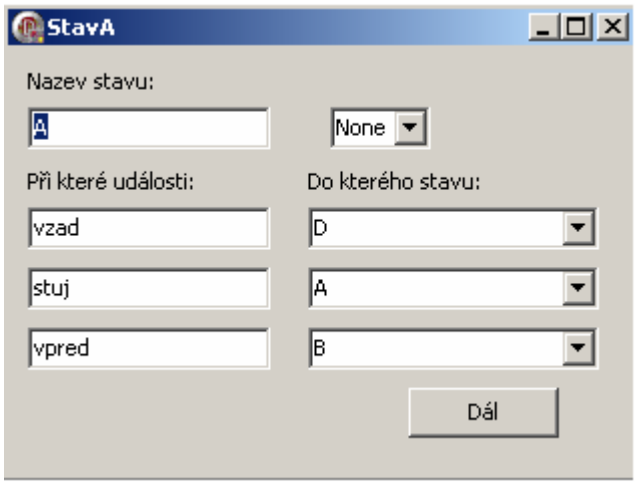


Obr. 6 - Automatické rozmístění stavů



Obr. 7 - Upravený stavový digram

Stavový diagram lze dále editovat opět pomocí přechodové funkce (rozmístění stavů zůstává zachováno) nebo lze měnit jednotlivé stavy individuálně pomocí menu, které se zobrazí po stisku pravého tlačítka myši na příslušném stavu. Menu umožňuje stav smazat, změnit jeho název nebo status nebo upravit cílové stavy jednotlivých přechodů. Při změně názvu stavu je opět testováno, zda jde o platný název.

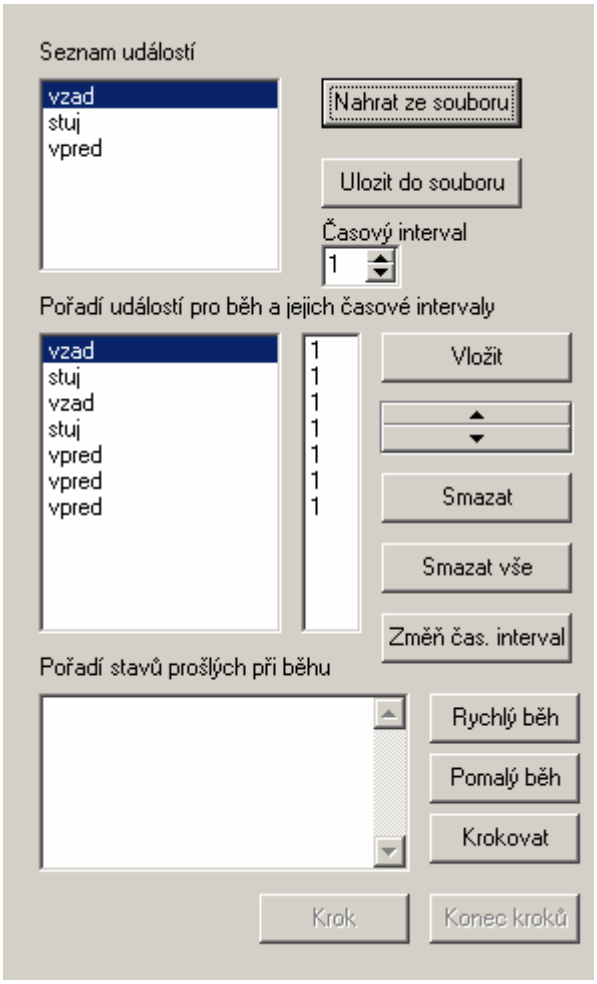


Obr. 8 - Menu pro editaci jednotlivých stavů

Hotový stavový diagram je možné uložit nebo ho použít k simulaci.

2.2 Simulace stavových digramů

Po stisku položky *Běh* v menu *Běh* se vytvoří panel, který zpřístupňuje všechny funkce potřebné pro simulaci jednoho stavového diagramu.

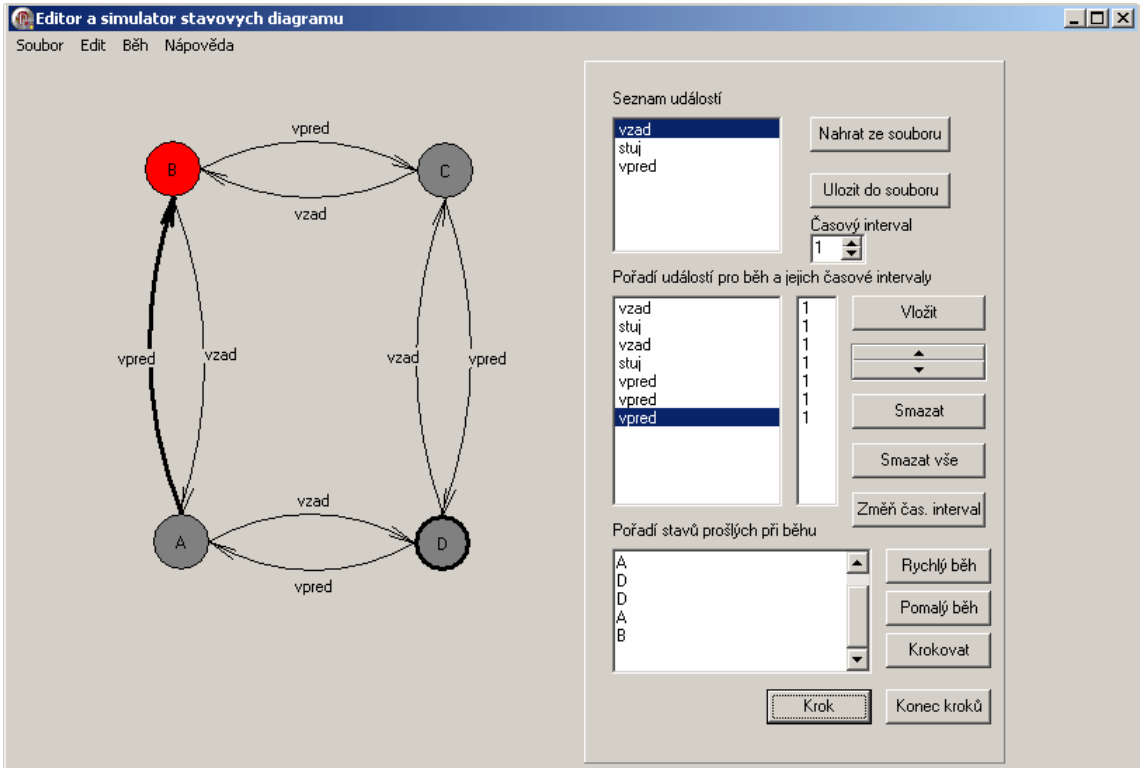


Obr. 9 - Panel pro simulaci

V *Seznamu událostí* jsou vypsány všechny povolené události pro daný stavový diagram. V něm lze označit událost, kterou chceme vložit do *Pořadí událostí pro běh*. To se provede po stisku tlačítka *Vložit*. S označenou událostí se do posloupnosti vloží také časový interval, který se nastavuje příslušným SpinEditem. Časové intervaly mají svůj účel pouze při simulaci více diagramů najednou. Pomocí tlačítek se šipkami lze měnit pořadí událostí, ve kterých budou vyvolávat přechody. Tlačítkem *Smazat* je možno odstranit událost označenou v *Pořadí událostí pro běh*. Po stisku tlačítka *Smazat vše* dojde k vymazání všech událostí i časových intervalů. Tlačítko *Změň čas. interval* nahradí zvolený časový interval hodnotou SpinEditu. Výstupem provedené simulace je seznam stavů, jak jimi bylo procházeno. Tento seznam se zobrazí v *Pořadí stavů prošlých při běhu*. *Seznam událostí* a *Pořadí událostí pro běh* i s časovými intervaly lze ukládat nebo nahrát.

Program nabízí tři možnosti simulace: rychlou, pomalou a krokování. Ty se spouští příslušnými tlačítky. Při rychlé simulaci se pouze vypíší stavy, jak jimi bylo projito. Při pomalé simulaci je běh znázorněn jednoduchou animací měnících se stavů. Při krokování se mění stavy po jednom po každém stisknutí tlačítka *Krok*, při čemž je vždy zvýrazněna cesta do aktuálního stavu.

Při spuštění jakéhokoli běhu se testuje zda jsou vytvořeny stavy, není prázdné *Pořadí událostí pro běh* a zda existuje vstupní stav ve stavovém diagramu. Poté se simulace provede. Na jejím konci se zobrazí hlášení, zda diagram skončil ve výstupním stavu. Výstupní stav je označen silnější čarou na obvodu. Červeně je zobrazen aktuální stav, ostatní jsou šedé. Aktuální stav na začátku simulace je vstupní.



Obr. 10 - Příklad simulace

Panel pro simulaci se uzavře položkou *Konec běhu* v menu *Běh*. Po uzavření se uloží pořadí událostí a časové intervaly do paměti a při opětovném vytvoření panelu se automaticky nahrají.

2.3 Běh více diagramů

Program umožňuje simulovat několik stavových diagramů najednou s jejich vzájemnou komunikací. Po stisku položky *Počet diagramů* v menu *Běh→Běh více diagramů* se zobrazí okno, v němž se zadává počet diagramů, jež chceme simulovat. Po zadání požadované hodnoty se zobrazí tento počet formulářů. V každém z nich lze vytvářet, upravovat a simulovat stavové diagramy nezávisle na sobě. Po stisku položky *Běh* v menu *Běh→Běh více diagramů* se ve všech formulářích vytvoří panely pro simulaci, ale kromě panelu v 1. diagramu nemají tlačítka pro spouštění běhu. Simulace více diagramů je tedy spouštěna z 1. diagramu.

Komunikace mezi diagramy je řešena přes komunikační funkce. Každý diagram má vlastní komunikační funkci, která se upravuje pomocí položky *Komunikační funkce* v menu *Edit*.

Obr. 11 - Příklad komunikační funkce

Tato tabulka má v prvním sloupci stavy stavového diagramu, ze kterého byla komunikační funkce zavolána. V následujících dvou sloupcích jsou rolovací menu obsahujících události jiného stavového diagramu a SpinEdity pro zadávání časových intervalů. Tyto dva sloupce se opakují pro všechny ostatní diagramy. Zadává se zde jaká událost a s jakým intervalem se má zařadit na začátek fronty v některém z jiných diagramů při příchodu do daného stavu. V uvedeném příkladě se při příchodu 1. diagramu do stavu A zařadí událost DoGamy s intervalem 2 na vrcholek fronty událostí v 3. diagramu. Všechny rolovací menu mají možnost prázdné události, pro případ, že po příchodu do nějakého stavu nechceme ovlivňovat všechny diagramy. Tato možnost je navíc implicitně nastavena všude, protože předpokládám, že posílání událostí bude mnohem méně častější než neposílání. Komunikační funkce je také možno ukládat a nahrávat.

Díky této komunikaci je možno simulovat situace, kdy jeden stavový diagram čeká, až v jiném diagramu nastane určitý stav apod..

V běhu více diagramů hrají důležitou roli časové intervaly. Označují, za jak dlouho od minulé události se spustí nová událost. Časový interval nemůže být nulový, protože není možné, aby v jednom diagramu došlo ke dvěma událostem naráz. To by vedlo k nejednoznačnosti. V běhu jednoho diagramu nezávisí na tom s jakými intervaly se události vykonávají, ale pouze na jejich pořadí. U více diagramů je nutné vědět, že například v 2. diagramu dojde k události dříve než v 1., protože tato událost může vyvolat událost, která se zařadí do fronty v 1. diagramu před událost, jež měla být právě vykonána. Je pravděpodobné, že v simulaci dojde k situaci, že by mělo najednou nastat

několik událostí, pak je program nastaven tak, že vyšší prioritu má událost v diagramu s nižším pořadovým číslem. Pokud tato událost vyvolá nějakou jinou, ta se může zařadit i před tu, která měla právě nastat.

I v tomto módu jsou stejné tři možnosti simulace. Fungují obdobně jako při jednoduchém běhu. Jenom se před každým krokem spočítá, ve kterém diagramu má nejdříve dojít k nějaké události a tímto diagramem se program zabývá. Při rychlé simulaci to není patrné, ale při pomalé simulaci a krokování dochází k přepínání mezi diagramy. Výsledkem simulací jsou jednotlivé seznamy stavů, jak jimi bylo projito, s oznámením každého stavového diagramu zda skončil ve výstupním stavu. Navíc je u každého stavu napsán časový údaj, kdy bylo tohoto stavu dosaženo, díky tomu je možné jasně vysledovat, jak za sebou události následovaly. Žádnou simulaci opět nelze spustit, pokud nejsou vytvořeny všechny stavové diagramy, nejsou zadány všechny posloupnosti událostí nebo chybí vstupní stav v některém stavovém diagramu.

Položka *Konec běhu* v menu *Běh→Běh více diagramů* pozavírá všechny běhové panely.

3 Implementace programu

3.1 Reprezentace součástí stavového diagramu

Jednotlivé stavy jsou tvořeny představiteli třídy TState1, kterou jsem pro tento účel vytvořil. Samotná třída je popsána níže. Stavy jsou umístěny na formuláři, který je překryt komponentou třídy TImage. Vlastnost Canvas této komponenty využívám k vykreslování jednotlivých přechodů. Je možné kreslit i přímo na formulář, ale pokud je takový obraz překryt jiným objektem (oknem, panelem, stavem,...), dojde k jeho smazání. Oproti tomu obraz u třídy TImage je uložen v paměti a tento problém tudíž nehrozí. Události vyvolávající jednotlivé přechody jsou zobrazeny jako instance třídy TLabel, která přesně splňuje požadavky pro jednoduchý popisek. Na panelu pro běh událostí je pak seznam událostí, jejich posloupnost i posloupnost časových intervalů představována seznamem třídy TListbox, která umožňuje jednoduchou manipulaci se svými položkami. Seznam prošlých stavů je tvořen komponentou třídy TMemo, což je rozšířené textové pole.

Deklarace všech níže popisovaných tříd, funkcí a procedur jsou umístěny pro přehlednost v přílohách.

3.2 Třída TState1

Důležitým stavebním kamenem programu je třída TState1, jejíž instance představují stavy v programu. Tato třída je odvozena od třídy TShape, která umožňuje nastavit kruhový tvar objektu, měnit jeho barvu a tloušťku okraje.

Navíc jsem této třídě přidal vlastnost Title, kde je uložen název příslušného stavu. Tento název musí být zobrazen. K tomu využívám instanci třídy TLabel. Třída TState1 stejně jako TShape nemůže obsahovat žádný jiný objekt, není tedy možné přiřadit Label stavu, který by se o něj automaticky staral, tzn. Label by byl vždy viditelný na stavu a pohyboval by se zároveň s ním. Toto jsem musel řešit jinak. Vytvořil jsem několik metod třídy TState1, které vytváří daný Label, umožňují změnu jeho textu a upravují jeho polohu, aby byl vždy přibližně ve středu stavu a byl správně zobrazen. To, že Label není součástí stavu, může být výhodné v okamžiku, kdy název stavu je příliš dlouhý a přesahuje hranice stavu. V tomto případě bude Label jednoduše

větší než stav a název bude i vně stavu. Pokud by Label byl součástí stavu, toto by nebylo možné.

Další vlastnost, kterou jsem třídě TState1 přidal je EventPath. Tato vlastnost v podstatě představuje individuální tabulku přechodů příslušného stavu. Jde o dynamické pole záznamů, které mají dvě položky. Položka Event je typu string a obsahuje název jedné z povolených událostí. Položka Target je typu integer a skrývá index stavu, do kterého vede přechod vyvolaný událostí v Event. Vlastnost EventPath jakožto dynamické pole vyžaduje také metody na nastavování a zjišťování délky pole. Tyto metody se nazývají SetL a GetL.

Jelikož jsou i stavy vytvořeny jako dynamické pole, je někdy potřeba zjistit index příslušného stavu (například při zjišťování, který stav volá obsluhu svého menu), k tomu využívám vlastnost Tag, do které daný index ukládám.

Aby bylo možné stavům přiřadit vlastní menu, musel jsem vlastnost PopupMenu zveřejnit přeřazením do sekce published. Třída TShape totiž tuto vlastnost nemá mezi zveřejněnými.

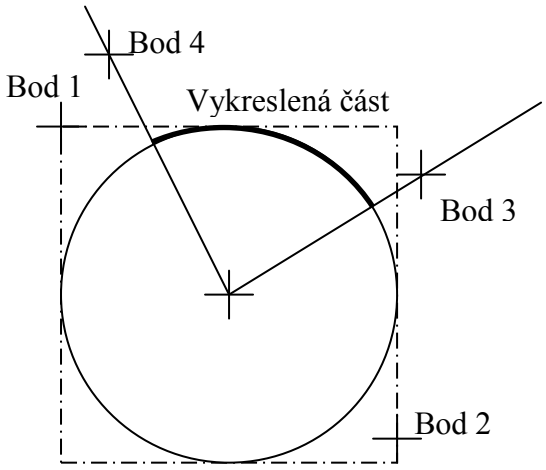
Poslední vlastností implementovanou do třídy TState1 je Status. Jde o výčtový typ s možnostmi sNone, sIO, sIN a sOUT, které reprezentují možné statusy stavu (ty byly popsány výše).

3.3 Vykreslování přechodů

Pro vykreslování stavových trajektorií používám proceduru ConnectAllStates, která se stará o vykreslení všech trajektorií a správný výpis událostí vyvolávajících dané přechody.

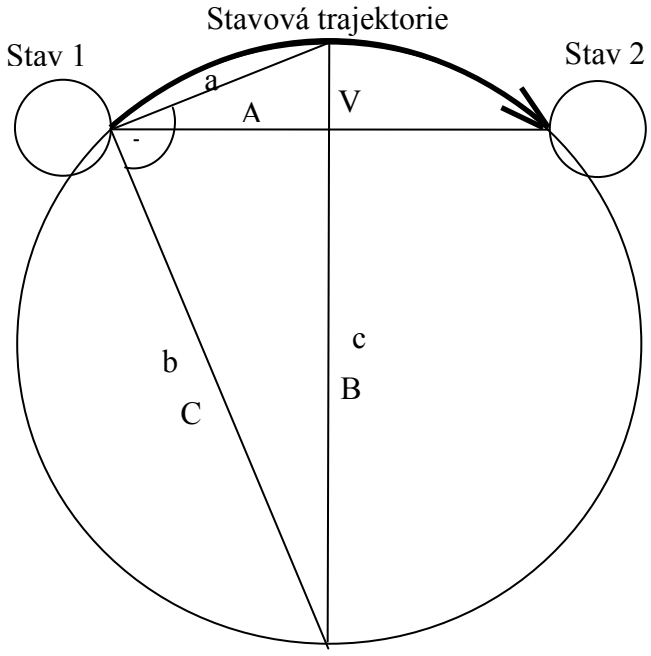
Samotné kreslení jednotlivých přechodů zajišťuje procedura ConnectStates, ve které jsou vypočítávány všechny body potřebné pro zobrazení přechodu. Vstupními parametry jsou dva stavy, které je třeba propojit a Label, ve kterém jsou napsány události vyvolávající tento přechod. První stav je původní a druhý cílový. Grafické znázornění přechodů provádím pomocí částí kružnic, na jednom konci zakončených šipkou, zobrazených mezi stavy. Pro vykreslení části kružnice v Delphi jsou potřeba souřadnice čtyř bodů. Dva protilehlé vrcholy pomyslného čtverce (body 1 a 2), ve kterém je kružnice vepsána a dva body určující úhel vykreslené části (body 3 a 4). Začátek vykreslené části se určí jako průsečík kružnice a polopřímky vedoucí ze středu

kružnice do bodu 3. Koncový bod se určí stejně jen s bodem 4. Směr vykreslování je proti směru hodinových ručiček.



Obr. 12 - Princip vykreslení části kružnice v Delphi

Vykreslený oblouk spojí body, kde jsou si stavy nejblíže a od spojnice stavů v jejím středu je vzdálen 20 bodů. Abych mohl vypočítat všechny potřebné body, musím zjistit velikost poloměru kružnice. K tomu mi pomůže vhodný nákres.



Obr. 13 - Geometrický nákres pro výpočet poloměru kružnice

Na obrázku jsou patrné tři pravoúhlé trojúhelníky: abc, ABC a AVa. Pomocí

$$a^2 + b^2 = c^2$$

Pythagorovy věty získáme rovnice: $A^2 + B^2 = C^2$

$$A^2 + V^2 = a^2$$

Z obrázku jsou patrné podmínky: $C = b \qquad B + V = c$

$$a^2 + C^2 = (B + V)^2$$

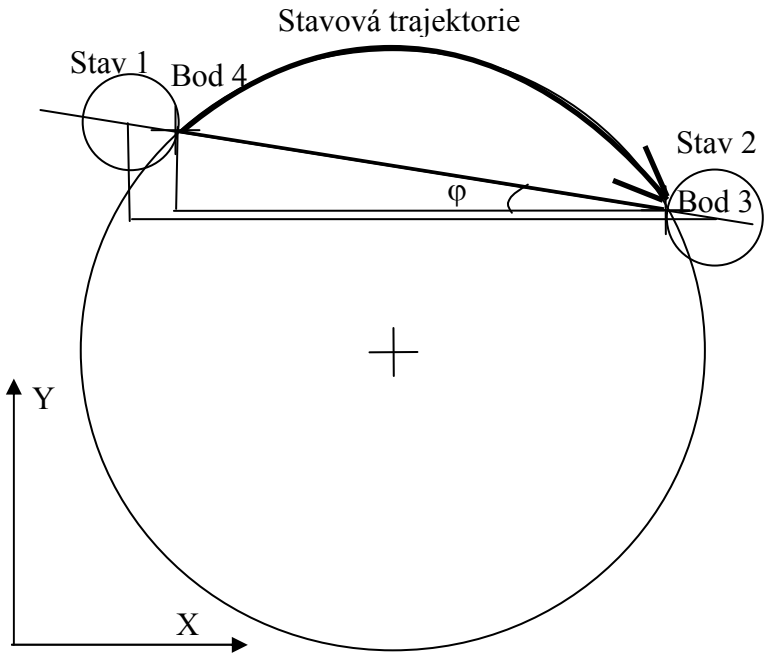
Po dosazení podmínek do rovnic získáme: $A^2 + B^2 = C^2$

$$A^2 + V^2 = a^2$$

Výsledkem dosazení druhé a třetí rovnice do první je: $A^2 + V^2 + A^2 + B^2 = B^2 + 2BV + V^2$

Po úpravě: $2A^2 = 2BV$ Po další úpravě: $B = \frac{A^2}{V}$

A je poloviční vzdálenost nejkratší spojnice mezi stavy, V jsem určil rovno 20 a B+V je pak průměr kružnice. A neznám a musím ho tedy vypočítat.



Obr. 14 - Obecná poloha stavů

Úhel φ mohu díky podobnosti trojúhelníků vypočítat jako:

$$arctg\frac{Stred2.Y - Stred1.Y}{Stred2.X - Stred1.X} = arctg\frac{Stav2.Top - Stav1.Top}{Stav2.Left - Stav1.Left}$$

Pokud by jmenovatel byl 0, pak úhel nastavím na π/2.

Souřadnice bodu 3 jsou pak:

X =Stav2.Left+20-round(Sign(Stav2.Left-Stav1.Left)*20*cos(φ));

Y = Stav2.Top+20-round(Sign(Stav2.Left-Stav1.Left)*20*sin(φ));

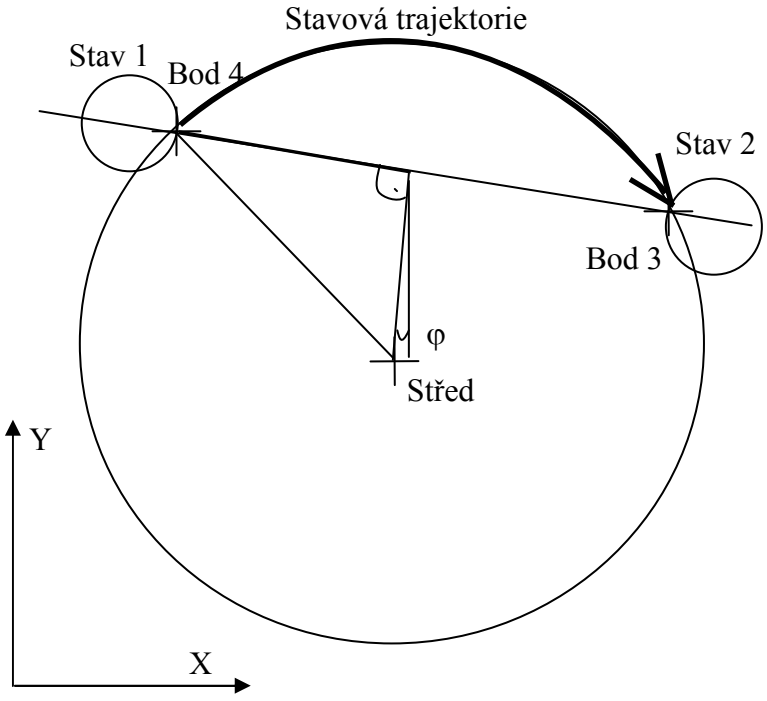
Bod 4 má tyto souřadnice:

X =Stav1.Left+20+round(Sign(Stav2.Left-Stav1.Left)*20*cos(φ));

Y = Stav1.Top+20+round(Sign(Stav2.Left-Stav1.Left)*20*sin(φ));

Funkce signum zde řeší i možnost, kdy by byly stavy prohozeny.

A^2 nyní snadno získám jako čtvrtinu kvadrátu vzdálenosti bodů 3 a 4. Po té již můžu vypočítat průměr potažmo poloměr kružnice. Abych mohl určit body 1 a 2, potřebuji ještě znát souřadnice středu kružnice.



Obr. 15 - Geometrické znázornění pro výpočet středu

Úhel φ je totožný s tím, který jsem vypočítal dříve. Souřadnice středu se vypočítají podle vzorců:

X = Stav1.Left+20+Sign(Stav2.Left-Stav1.Left)*round(20*cos(φ)+A*cos(φ)-(r-20)* sin φ));

Y=Stav1.Top+20+Sign(Stav2.Left-Stav1.Left)*round(20*sin(φ)+A*sin(φ)+(r-20)*cos(φ));

Body 1 a 2 se vypočítají tak, že ke středu připočtu nebo odečtu poloměr. Nyní již znám všechny potřebné body a mohu nakreslit požadovaný oblouk. Abych mohl nakresli v bodě 3 šipku, musím vědět pod jakým úhlem do něj míří oblouk. Úhel spočítám jako: $\alpha = \frac{\pi}{2} - arctg \frac{Stred.Y - Bod3.Y}{Bod3.X - Stred.X}$

K úhlu přidám 5° směrem ke středu a 15° od středu a pod těmito úhly nakreslím z bodu 3 dvě úsečky. Směr kreslení určí signum rozdílu Bod3.X-Stred.X.

Souřadnice popisky stavového diagramu je určena tak aby její střed byl 10 bodů nad vrcholem trajektorie.

Procedura ConnectAllStates si vede tabulku stavu, kterým se zabývá, ve které jsou zapsány stavy, do kterých už byl vykreslen stav. Pokud se má vytvořit přechod do stavu, do kterého už vykreslen byl, procedura zařídí, aby se do popisky přidala nová událost a upravila se její poloha.

3.4 Zobrazení dynamiky přechodů

Při krokování a pomalé simulaci dochází ke zvýraznění cesty z jednoho stavu do druhého. Při krokování naráz a v pomalé simulaci postupně. Jde o velmi jednoduchou animaci průběhu. K této funkci byla vytvořena funkce ShowPath, která funguje velmi podobně jako ConectStates. První dva parametry jsou stejné, třetí parametr je ale v tomto případě číslo části křivky, kterou chceme zvýraznit. S popiskou přechodu nepotřebujeme nic dělat. Pokud je třetí parametr 0, dojde k vykreslení celé trajektorie, při hodnotách 1, 2 nebo 3 dojde k vykreslení její příslušné části. Hodnota 0 je použita při krokování, zbývající tři při pomalém běhu. Pokud je hodnota větší jak 0, je třeba spočítat ještě dva další body navíc k těm, které se počítali v ConectStates. Tyto body mají rozdělit úhel mezi body 3 a 4 na třetiny. Jejich výpočet je jednoduchý. X-ové souřadnice získám jako rozdíl x-ových sořadnic bodů 3 a 4 vynásobený 1/3 respektive 2/3 a přičtený k bodu 4. Stejně získám i y-ové souřadnice. Vykreslování v této funkci je prováděno o dva pixely tlustší čarou než v ConnectStates.

3.5 Zjišťování stavu, do kterého se má stavový diagram přesunout

Pro tento účel jsem vytvořil funkci NextState, která má jako vstupní parametry jméno právě nastalé události a index aktuálního stavu. Návratová hodnota funkce je index nového stavu. Funkce plně využívá vlastnosti třídy TState1 EventPath. Tato

vlastnost aktuálního stavu je cyklicky procházena dokud se položka Event neshoduje s hledanou událostí, poté je odpovídající položka Target použita jako návratová hodnota.

3.6 Komunikace mezi diagramy

Rozesílání událostí ostatním diagramům zařizuje procedura ComSD. Její vstupní parametr je index aktuálního stavu. Procedura na začátku zjišťuje, který diagram ji zavolal. Pokud je volajícím Form1, tak projede všechny ostatní diagramy, které jsou v dynamickém poli Diags a podle vyplněné komunikační funkce zařazuje události a časové intervaly na pozici před ukazovátkem příslušných seznamů, ukazovátka jsou pak nasměrována na ně. Pokud byl některý stavový digram, do kterého má být přidána událost, označen příznakem Kon (už byly provedeny všechny události v seznamu), jsou událost a časová interval zařazeny až na konec seznamů, ukazovátka jsou přesunuta na ně a příznak Kon je zrušen. V případě, že volající není Form1, jsou projity všechny ostatní členy Diags a nakonec i Form1 a jsou provedeny stejné kroky jako v předešlém případě. Ve všech diagramech, do kterých byla přidána událost se nastaví stejný čas jako v diagramu, který ComSD zavolal, tím dojde ke synchronizaci.

3.7 Průběh jednoho kroku

Provedení jednoho kroku ve stavovém diagramu zajišťuje procedura NextStep pro rychlou simulaci a krokování a NextStepOT pro pomalou simulaci. Obě funkce provádějí v podstatě totéž, ale NextStapeOT v několika krocích. NextStape má vstupní parametr, který může nabýt dvou hodnot, kterými se odlišuje rychlá simulace od krokování. U NextStapeOT vstupní parametr určuje fázi vykreslování a může nabývat tří hodnot. Obě funkce nejprve zavolají funkci NextState, aby zjistili nový stav, pokud je nový rozdílný od starého, starý se přebarví na šedo a nový na červeno, pokud simulace není rychlá, tak se zavolají procedury ConnectAllStates a ShowPath. Vždy se vypíše nový stav a při komunikaci více diagramů i aktuální čas. Následně se do aktuálního stavu přiřadí nový a zkontroluje, zda program došel již nakonec posloupnosti. Pokud ne, posune se ukazovátko na další událost a časový interval v seznamu, jinak se nastaví příznak Kon stavového diagramu. Při běhu více diagramů je na konci zavolána procedura ComSD.

3.8 Určování aktivního stavového diagramu

Pro určení stavového diagramu, ve kterém se má vyhodnotit nějaká událost, má každý diagram vlastnost TimeValue, ve které je uložena aktuální hodnota času v tomto diagramu. Na začátku každého kroku se k této hodnotě připočte časový interval události, která je v příslušném diagramu na řadě. Který diagram má tuto hodnotu nejnižší, v tom se událost vyhodnotí. Hodnota ve Form1 se na začátku vždy nastaví jako nejnižší a pak jsou s ní porovnávány ostatní hodnoty. Porovnávají se však pouze diagramy bez nastaveného příznaku Kon. Pokud však má Form1 nastaven příznak Kon, je opět jeho hodnota nastavena jako nejmenší, ale je zvětšena na 10000, takže první neukončený diagram bude brán jako nejmenší. Jakmile mají všechny diagramy nastavený příznak Kon, dojde k ukončení simulace.

3.9 Datové formáty

3.9.1 Úvod

Pro ukládání potřebných dat do souborů jsou navrženy čtyři datové formáty. Navrhoval jsem je tak, aby v nich byly všechny důležité hodnoty, ale zároveň soubory nebyly zbytečně velké. Všechny čísla mají velikost 1B, tzn. že jsou omezeny hodnotou 255, což všude kromě souřadnic stavů dostatečně vyhovuje. Aplikace umožňuje vytvořit stavový diagram s více jak 255 stavy nebo událostí, ale takový stavový diagram by byl značně nepřehledný a neměl by žádný praktický smysl. Souřadnice stavů jsou ukládány jako dvě čísla, pomocí nichž se vypočte skutečná hodnota.

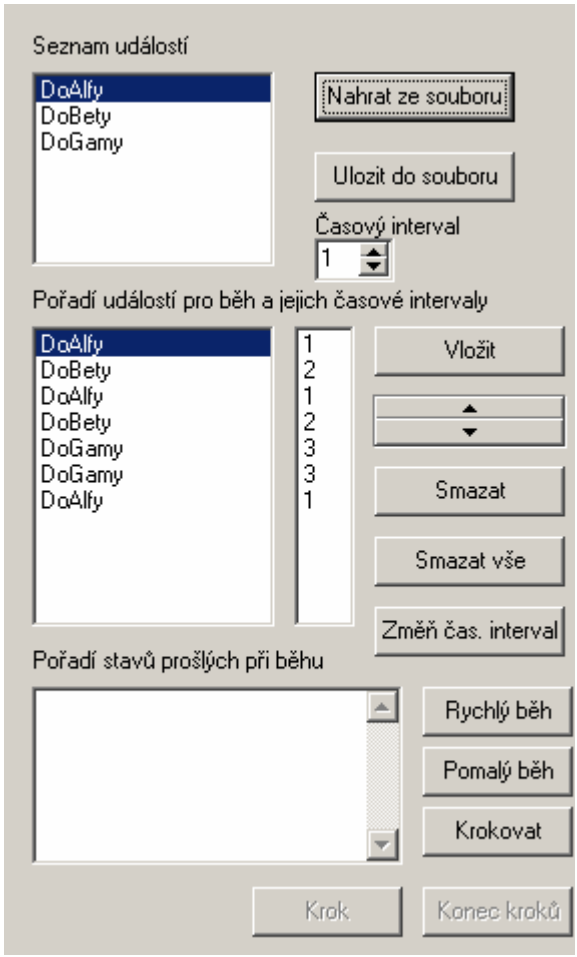
3.9.2 Formát pfc

Pro ukládání informací z přechodové funkce je navržen formát pfc. Informace jsou v něm uloženy v pořadí: počet událostí, seznam událostí jako řetězce, jednotlivé položky oddělené středníkem, dále počet stavů, název prvního stavu jako řetězec ukončený středníkem, index jeho statusu (0 pro neutrální stav, 1 pro vstup/výstup, 2 pro vstup, 3 pro výstup) a indexy stavů, do kterých vedou přechody pro události v pořadí seznamu, stejně i pro ostatní stavy.

3.9.3 Formát evo

Pro ukládání informací o posloupnosti událostí je navržen formát evo. Informace jsou v něm uloženy v pořadí: počet událostí, seznam událostí jako řetězce, jednotlivé položky oddělené středníkem, poté index první události v pořadí odpovídající jejímu umístění v seznamu událostí, dále jí odpovídající časový interval, stejně pro další události v pořadí.

Pro příklad zvolím takovouto posloupnost událostí:



Obr. 16 – Příklad pro ukládání posloupností událostí

Takový to diagram se uloží v tomto formátu:

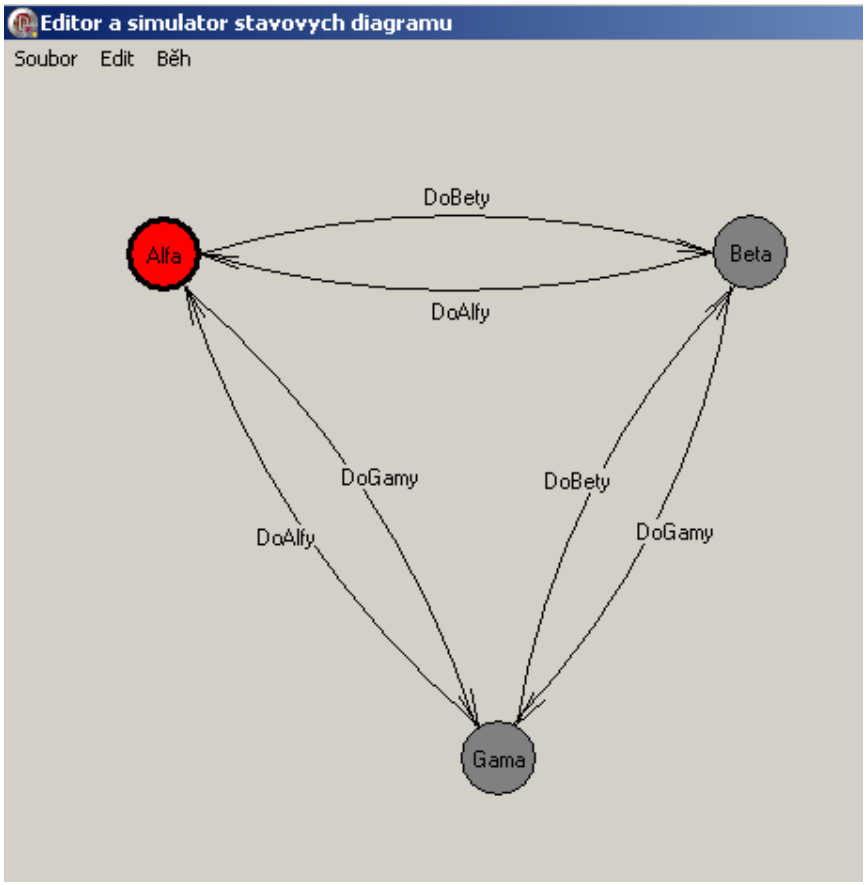
3DoAlfy;DoBety;DoGamy;0 1 1 2 0 1 1 2 2 3 3 0 1

Jednotlivá čísla jsou ve skutečnosti nahrazeny znak s jim odpovídající ASCII hodnotou a není mezi nimi mezera. Tato reprezentace čísel platí pro všechny formáty.

3.9.4 Formát sts

Pro ukládání informací o stavech je navržen formát sts. Informace jsou v něm uloženy v pořadí: počet událostí, seznam událostí jako řetězce, jednotlivé položky oddělené středníkem, dále počet stavů, název prvního stavu jako řetězec ukončený středníkem, číslo i a číslo j z nichž se určí x-ová souřadnice stavu x = 255*i+j, z dalších dvou čísel se určí y-ová souřadnice, následuje index stavového statusu (0 pro neutrální stav, 1 pro vstup/výstup, 2 pro vstup, 3 pro výstup) a indexy stavů, do kterých vedou přechody pro události v pořadí seznamu, stejně i pro ostatní stavy.

Pro příklad zvolím takovýto stavový diagram:



Obr. 17 – Příklad pro ukládání stavových diagramů

Souřadnice stavu Alfa jsou: Left = 65 Top = 74

Souřadnice stavu Beta jsou: Left = 377 Top = 73

Souřadnice stavu Gama jsou: Left = 243 Top = 307

Takový to diagram se uloží v tomto formátu:

3DoAlfy;DoBety;DoGamy;3Alfa;0 65 0 74 2 0 1 2Beta;1 122 0 73 0 0 1 2Gama; 243 2 87 0 0 1 2

3.9.5 Formát kfc

Pro ukládání informací z komunikační funkce je navržen formát kfc. Informace jsou v něm uloženy v pořadí: index události, která se bude posílat do prvního jiného diagramu po příchodu do prvního stavu (indexováno od 1, 0 znamená žádná událost), hodnota časového intervalu pro tuto událost, to samé pro další stavy až do posledního, pak vše pro ostatní diagramy.

4 Ověření funkce aplikace na konkrétních stavových diagramech

4.1 TCP spojení

4.1.1 Stavy TCP

Pro ověření funkčnosti aplikace jsem zvolil stavový diagram TCP spojení. TCP spojení se může nacházet v následujících stavech:

- CLOSED: spojení není navázáno.
- LISTEN: čekání na příchozí spojení (na straně serveru je port otevřen).
- SYN SENT: probíhá navazování nového spojení (na straně klienta zaslán SYN paket, od protistrany očekáváme potvrzení SYN-ACK).
- SYN RECV (=SYN RCVD): probíhá navazování nového spojení (na straně serveru jsme obdrželi SYN paket, odpověděli jsme zasláním SYN-ACK a očekáváme potvrzení)
- ESTABLISHED: spojení je navázáno a je plně funkční (připraveno k přenosu dat nebo přenos dat probíhá).
- FIN WAIT1: ukončení spojení inicializované naší stranou (protistraně jsme zaslali FIN paket a čekáme na jeho potvrzení).
- FIN WAIT2: pokračuje ukončení spojení inicializované naší stranou (obdrželi jsme potvrzení námi zaslaného FIN paketu a očekáváme FIN paket protistrany).
- TIME WAIT: poslední fáze námi inicializovaného ukončení spojení (obdrželi jsme FIN paket od protistrany a potvrdili jeho přijetí, po uplynutí prodlevy (která je v RFC definovaná jako dvojnásobek hodnoty MSL - "Maximum Segment Lifetime") přejdeme do stavu CLOSED).
- CLOSING: pokračuje ukončení spojení inicializované naší stranou (protistraně jsme zaslali FIN paket a očekávali jeho potvrzení. Mezitím jsme ale obdrželi od protistrany FIN paket, potvrdíme tedy jeho přijetí a dále čekáme na potvrzení námi zaslaného FIN).
- CLOSE WAIT: ukončení spojení inicializované protistranou (obdrželi jsme FIN paket a potvrdili jeho přijetí).
- LAST ACK: pokračuje ukončení spojení inicializované protistranou (poslali jsem FIN a čekáme na jeho potvrzení, poté přejdeme do stavu CLOSED).

4.1.2 Navazování spojení

Simuloval jsem několik průběhů navazování a ukončování spojení, proto je vhodné popsat jak takové navazování a ukončování spojení v TCP probíhá.

Při navazování TCP spojení se uplatňuje mechanismus tzv.trojnásobného potřesení rukou (three-way handshake). Během tohoto handshake se obě strany dohodnou na počátečních číslech sekvence:

- klient nejprve pošle serveru paket, který má nastavený SYN příznak (žádost o navázání spojení) a současně obsahuje prvotní číslo sekvence ze strany klienta (ISN "Initial Sequence Number" klienta). Na straně klienta se spojení nachází ve stavu SYN SENT.
- server po obdržení SYN paketu odpoví klientu paketem, který má nastavené příznaky SYN a ACK, dále obsahuje prvotní číslo sekvence ze strany serveru (ISN serveru) a potvrzené číslo sekvence klienta (ISN klienta zvýšené o 1). Na straně serveru se spojení nachází ve stavu SYN RECV.
- klient odpoví paketem s nastaveným příznakem ACK a potvrzením čísla sekvence serveru (ISN serveru zvýšené o 1). Jakmile paket dorazí k serveru, spojení se na obou stranách nachází ve stavu ESTABLISHED.

4.1.3 Ukončování spojení

Při ukončení TCP spojení si obě strany vymění pakety s nastaveným příznakem FIN a obě strany také potvrdí přijetí FIN paketu.

Z pohledu strany, která iniciuje ukončení spojení, to vypadá takto:

- pošleme FIN paket a čekáme na jeho potvrzení protistranou (FIN WAIT1);
- pak buď obdržíme potvrzení, že náš FIN paket byl přijat a čekáme, až protistrana bude připravena spojení ukončit a pošle svůj FIN paket (FIN WAIT2), jakmile FIN paket protistrany obdržíme, potvrdíme jeho přijetí a přejdeme do stavu TIME WAIT. Po uplynutí prodlevy dané specifikací TCP spojení zaniká (CLOSED);
- anebo k nám dříve než potvrzení našeho FIN paketu dorazí FIN paket protistrany, v tom případě přejdeme do stavu CLOSING, potvrdíme protistraně

přijetí FIN paketu a dále čekáme na potvrzení již dříve námi zaslaného FIN paketu. Jakmile jej obdržíme, přejdeme do stavu TIME_WAIT a po uplynutí prodlevy dané specifikací TCP spojení zaniká (CLOSED).

Z pohledu strany, která ukončení spojení neiniciovala, to vypadá takto:

- obdržíme FIN paket protistrany, potvrdíme jeho přijetí (CLOSE_WAIT);
- jakmile jsme připraveni spojení ukončit, pošleme náš FIN paket a čekáme na jeho potvrzení protistranou (LAST ACK);
- jakmile dorazí potvrzení, že byl náš FIN paket přijat, spojení zaniká (CLOSED).

Jak je vidět, tak na straně, která iniciuje ukončení spojení před jeho úplným zaniknutím určitou dobu čekáme ve stavu TIME WAIT. Tato prodleva existuje z toho důvodu, že se v síti mohou určitou dobu ještě vyskytovat pakety protistrany patřící k tomuto TCP spojení.

4.1.4 Události TCP

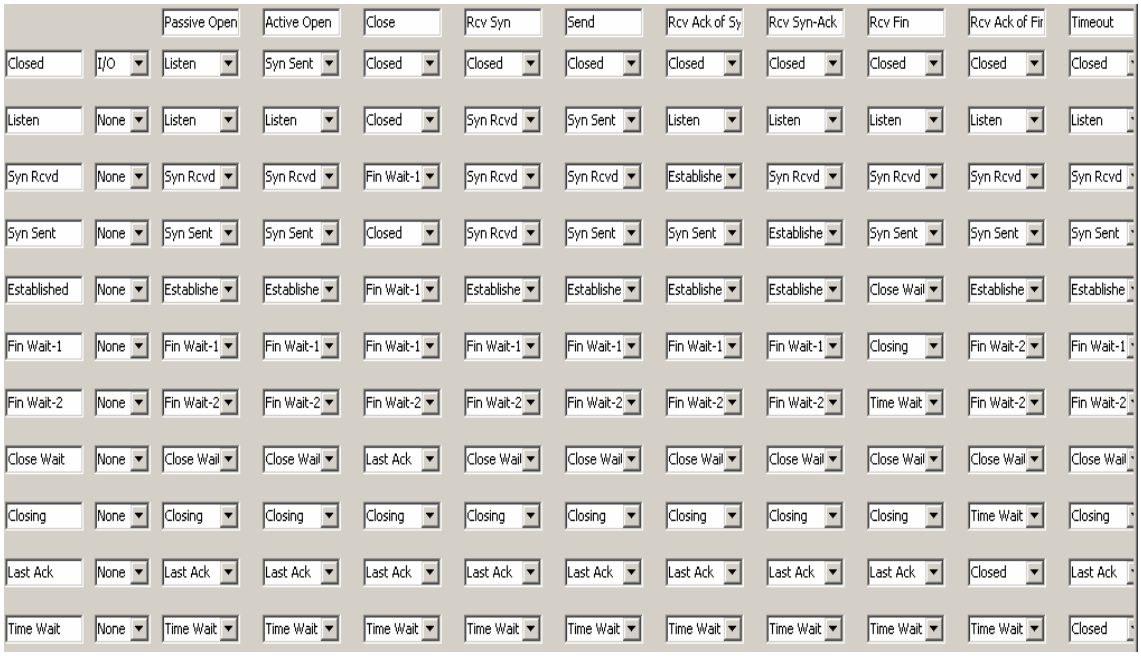
Během popisu navazování a ukončování spojení bylo již zmíněno několik událostí vyvolávajících přechody. Šlo převážně o obdržení paketu s nějakým nastaveným příznakem, existují ale i další události. Mezi ně ovšem naopak nepatří posílání různých paketů. Tyto akce jsou vyvolány buď některým z přechodů nebo příchodem do určitého stavu, pro náš pohled na stavový diagram však nehrají žádnou roli. Události při TCP spojení mohou být tyto:

- PASSIVE OPEN: umožnění navázání spojení (ale bez jeho inicializace).
- ACTIVE OPEN: aktivní pokus o navázání spojení (vyslán SYN paket).
- CLOSE: ukončení spojení (vyslán FIN paket).
- Rcv SYN: obdržení SYN paketu, protistrana chce navázat spojení (vyslán ACK of SYN paket).
- SEND: přechod z naslouchání k inicializaci spojení (ve stavu LISTEN byl vyslán SYN paket).
- Rcv ACK of SYN: obdržení potvrzení přijetí SYN paketu.
- Rcv SYN-ACK: obdržení SYN paketu i potvrzení přijetí SYN paketu.
- Rcv FIN: obdržení FIN paketu, protistrana chce ukončit spojení.
- Rcv ACK of FIN: obdržení potvrzení přijetí FIN paketu.

- TIMEOUT: vypršení doby 2*MSL.

4.1.5 Stavový diagram TCP

Přechodová funkce má podobu:

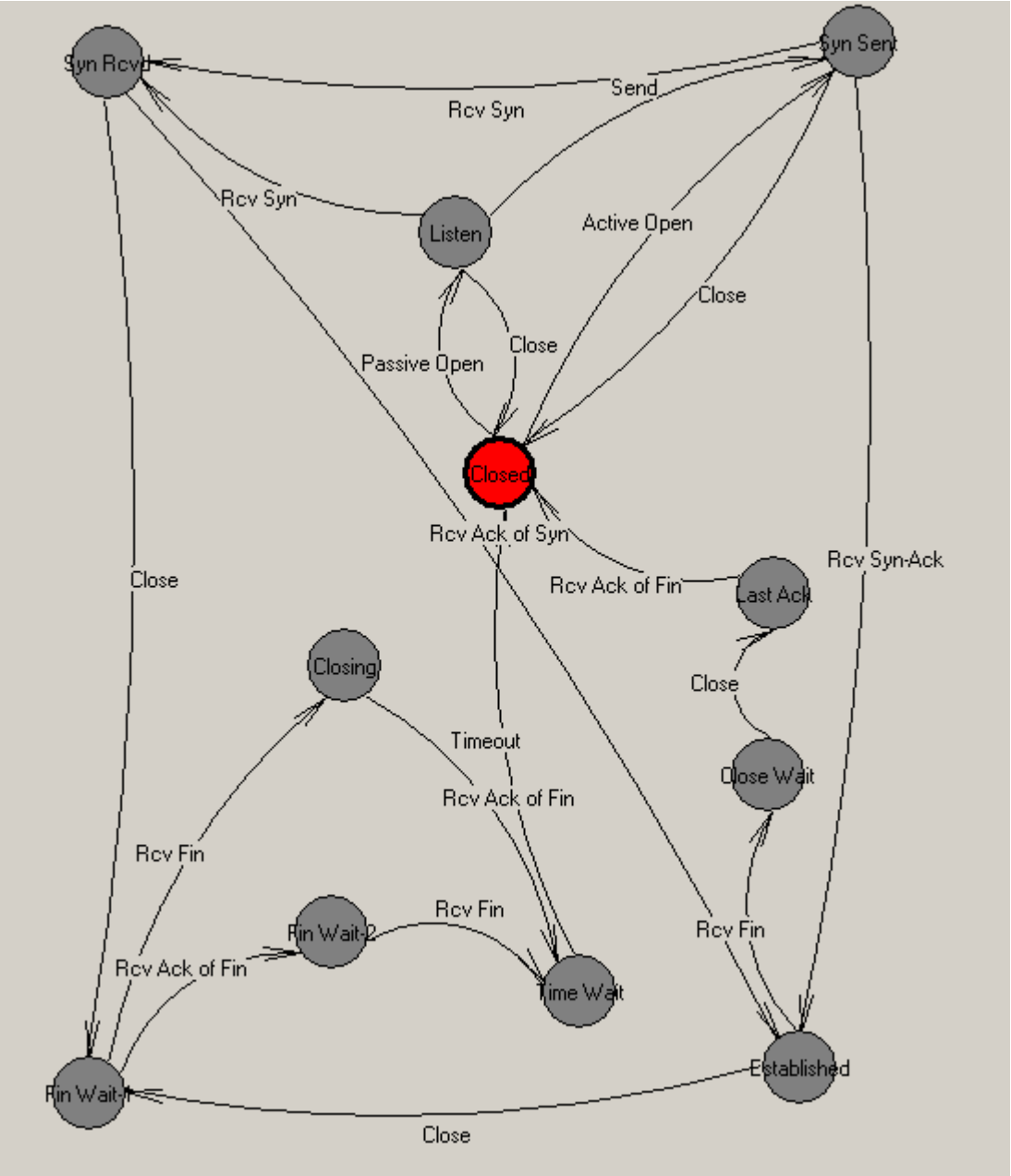


		Passive Open	Active Open	Close	Rcv Syn	Send	Rcv Ack of Sy	Rcv Syn-Ack	Rcv Fin	Rcv Ack of Fir	Timeout
Closed	I/O	Listen	Syn Sent	Closed	Closed	Closed	Closed	Closed	Closed	Closed	Closed
Listen	None	Listen	Listen	Closed	Syn Rcvd	Syn Sent	Listen	Listen	Listen	Listen	Listen
Syn Rcvd	None	Syn Rcvd	Syn Rcvd	Fin Wait-1	Syn Rcvd	Syn Rcvd	Establishe	Syn Rcvd	Syn Rcvd	Syn Rcvd	Syn Rcvd
Syn Sent	None	Syn Sent	Syn Sent	Closed	Syn Rcvd	Syn Sent	Syn Sent	Establishe	Syn Sent	Syn Sent	Syn Sent
Established	None	Establishe	Establishe	Fin Wait-1	Establishe	Establishe	Establishe	Establishe	Close Wail	Establishe	Establishe
Fin Wait-1	None	Fin Wait-1	Fin Wait-1	Fin Wait-1	Fin Wait-1	Fin Wait-1	Fin Wait-1	Fin Wait-1	Closing	Fin Wait-2	Fin Wait-1
Fin Wait-2	None	Fin Wait-2	Fin Wait-2	Fin Wait-2	Fin Wait-2	Fin Wait-2	Fin Wait-2	Fin Wait-2	Time Wait	Fin Wait-2	Fin Wait-2
Close Wait	None	Close Wail	Close Wail	Last Ack	Close Wail	Close Wail	Close Wail	Close Wail	Close Wail	Close Wail	Close Wail
Closing	None	Closing	Closing	Closing	Closing	Closing	Closing	Closing	Closing	Time Wait	Closing
Last Ack	None	Last Ack	Last Ack	Last Ack	Last Ack	Last Ack	Last Ack	Last Ack	Last Ack	Closed	Last Ack
Time Wait	None	Time Wait	Time Wait	Time Wait	Time Wait	Time Wait	Time Wait	Time Wait	Time Wait	Time Wait	Closed

Obr. 18 - Přechodová funkce TCP spojení

Je patrné, že většina stavů má definovány jen jeden nebo dva přechody, zde je vidět výhoda způsobu inicializace přechodů, kterou jsem použil. Jako vstupní i výstupní stav jsem zvolil stav Closed, ve kterém stav začíná i končí každou komunikaci.

Výsledný stavový diagram vypadá takto:



Obr. 19 - Stavový diagram TCP spojení

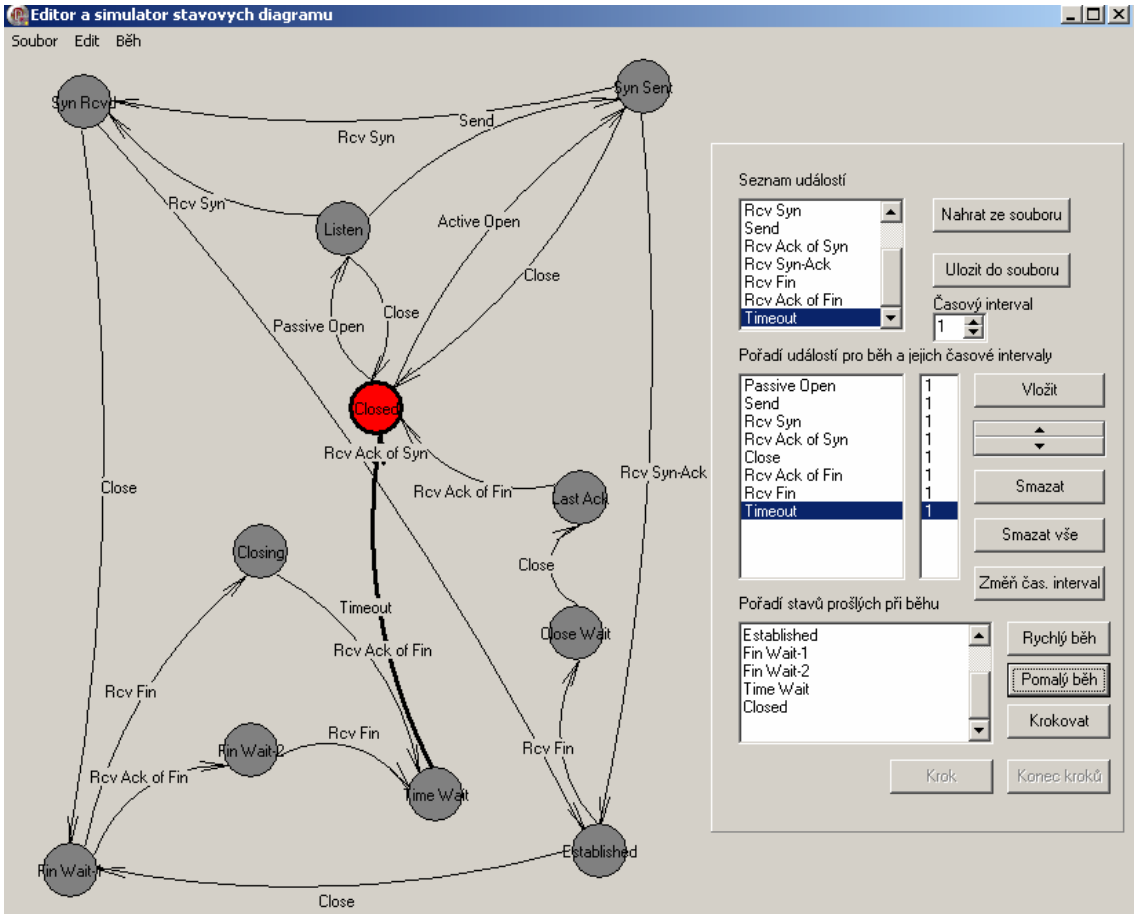
Při použití mého způsobu vykreslování stavových trajektorií se v tomto případě bohužel není možné vyhnout křížení některých trajektorií. Přechod ze stavu Syn Rcvd do Established by bylo třeba vést kolem stavu Fin Wait-1.

4.2 První simulace

V první simulaci jsem zadal posloupnost událostí: Passive Open, Send, Rcv Syn, Rcv Ack of Syn, Close, Rcv Ack of Fin, Rcv Fin a Timeout. Jde tedy o simulaci, kdy

spojení je uzavřeno, pak začneme naslouchat, vyšleme žádost o navázání spojení, přijmeme stejnou žádost od protistrany, přijmeme potvrzení naší žádosti, čímž se naváže spojení. Poté začneme ukončení spojení, obdržíme potvrzení, obdržíme žádost o ukončení i od protistrany a po uplynutí časové prodlevy dojde k ukončení spojení. Stavy by tedy měli po sobě následovat v pořadí: Closed, Listen, Syn Sent, Syn Rcvd, Established, Fin Wait-1, Fin Wait-2, Time Wait, Closed.

Výsledek první simulace:



Obr. 20 - První provedená simulace

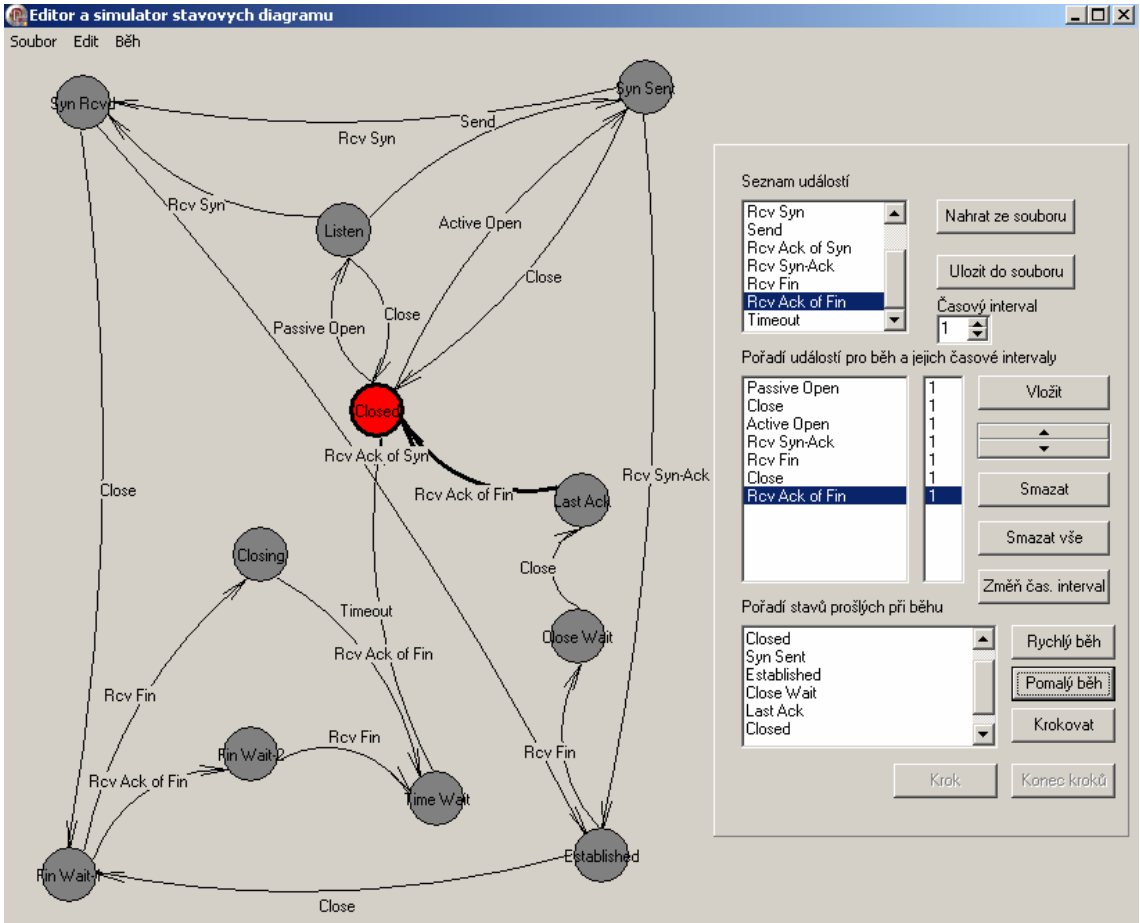
Prošlé stavy: Closed(počáteční stav není vypsán), Listen, Syn Sent, Syn Rcvd, Established, Fin Wait-1, Fin Wait-2, Time Wait, Closed. Bylo dosaženo výstupního stavu. Simulace tedy souhlasí s teoretickým předpokladem.

4.3 Druhá simulace

V druhé simulaci jsem určil pořadí událostí takto: Passive Open, Close, Active Open, Rcv Syn-Ack, Rcv Fin, Close, Rcv Ack of Fin. V simulaci je spojení uzavřeno, pak začneme naslouchat, opět spojení zavřeme, aktivně se pokusíme navázat spojení,

přijmeme žádost i potvrzení naší žádosti od protistrany, dojde k navázání spojení. Následně obdržíme žádost o ukončení spojení od protistrany, vyšleme souhlas s ukončením a obdržíme potvrzení, poté dojde k ukončení spojení. Stavy by tedy měli po sobě následovat v pořadí: Closed, Listen, Closed, Syn Sent, Established, Close Wait, Last-Ack, Closed.

Výsledek druhé simulace:



Obr. 21 - Druhá provedená simulace

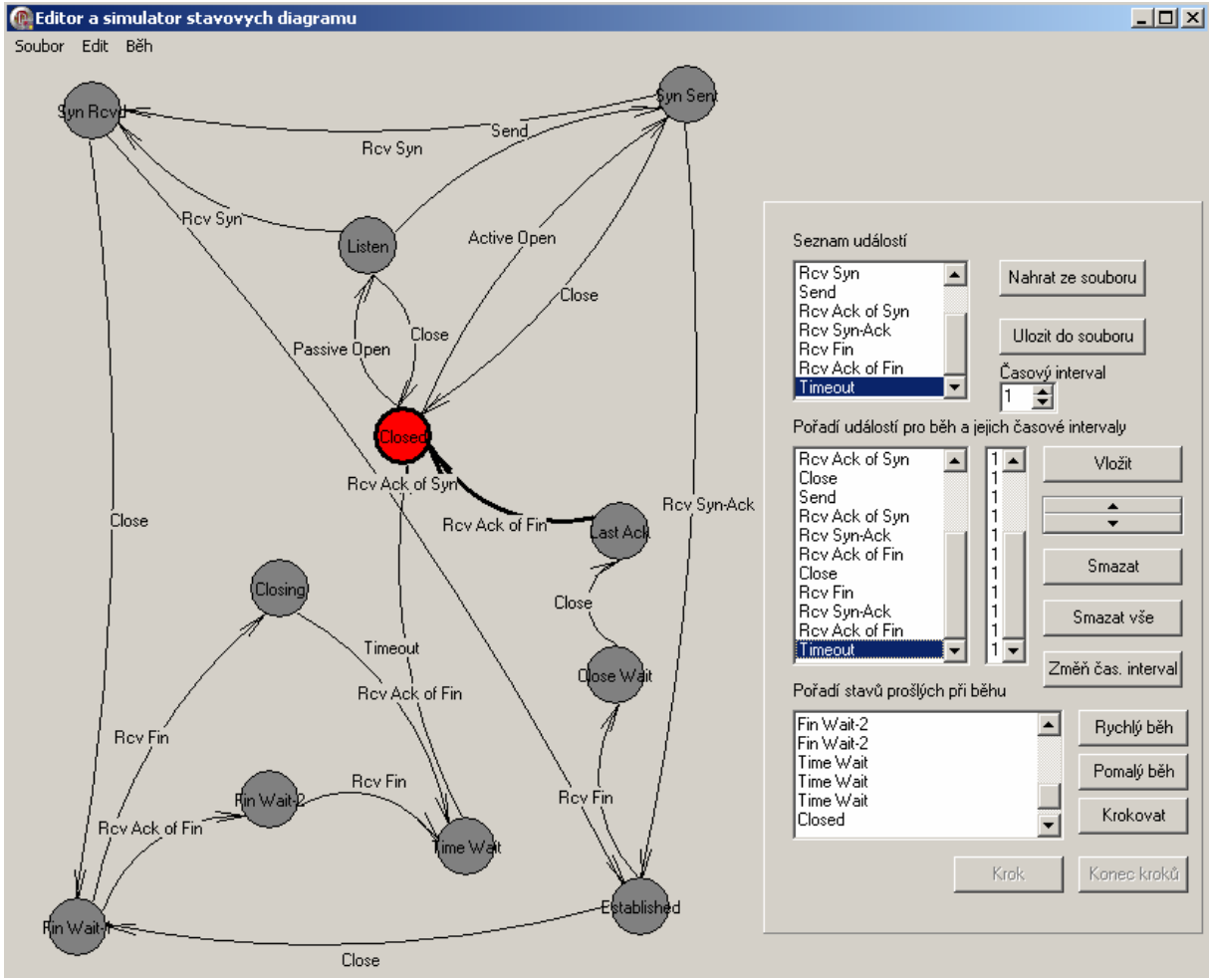
Prošlé stavy: Closed(počáteční stav není vypsán), Listen, Closed, Syn Sent, Established, Close Wait, Last Ack, Closed.. Bylo dosaženo výstupního stavu. Simulace opět souhlasí s teoretickým předpokladem.

4.4 Třetí simulace

Při třetí simulaci jsem zkoušel, chování systému při náhodné sekvenci událostí. Události jdou v pořadí: Passive Open, Active Open, Rcv Fin, Rcv Syn, Send, Passive Open, Rcv Ack of Syn, Close, Send, Rcv Ack of Syn, Rcv Syn-Ack, Rcv Ack of Fin,

Close, Rcv Fin, Rcv Syn-Ack, Rcv Ack of Fin, Timeout. Stavový diagram často narazí na přechod, který není pro daný stav definován měl by tak většinou zachovávat stav. Stavy by měli následovat: Closed, Listen, Listen, Listen, Syn Rcvd, Syn Rcvd, Syn Rcvd, Established, Fin Wait-1, Fin Wait-1, Fin Wait-1, Fin Wait-1, Fin Wait-2, Fin Wait-2, Time Wait, Time Wait, Time Wait,, Closed.

Výsledek třetí simulace:



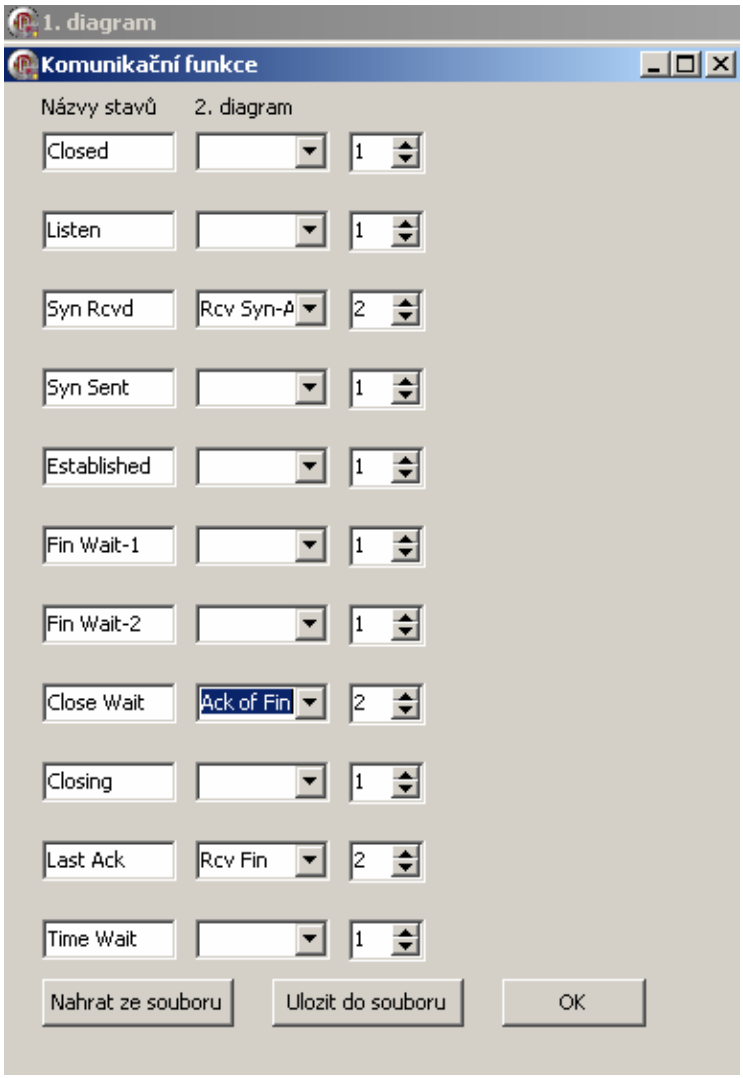
Obr. 22 - Třetí provedená simulace

Prošlé stavy: Closed(počáteční stav není vypsán), Listen, Listen, Listen, Syn Rcvd, Syn Rcvd, Syn Rcvd, Established, Fin Wait-1, Fin Wait-1, Fin Wait-1, Fin Wait-1, Fin Wait-2, Fin Wait-2, Time Wait, Time Wait, Time Wait, Closed. Simulace i v tomto případě souhlasí s teoretickým předpokladem.

4.5 Čtvrtá simulace

Čtvrtá simulace se týkala komunikace dvou stran TCP. Události v prvním diagramu následují v pořadí: Passive Open s časovým intervalem 1 a Close s intervalem 10. Toto pořadí znamená, že tato strana začne naslouchat a pokud se 10 časových jednotek nebude nic dít, tak spojení uzavře. V druhém diagramu jsem posloupnost událostí nastavil takto: Active Open s intervalem 2, Close s intervalem 5 a Timeout s intervalem 10. Tato strana tedy vyvolá spojení, po jeho navázání bude 5 časových jednotek komunikovat a pak ukončí spojení. Po vypršení 10 časových jednotek dojde k Timeoutu. Tyto krátké posloupnosti budou během simulace doplňovány navzájem posílanými událostmi, přes komunikační funkce.

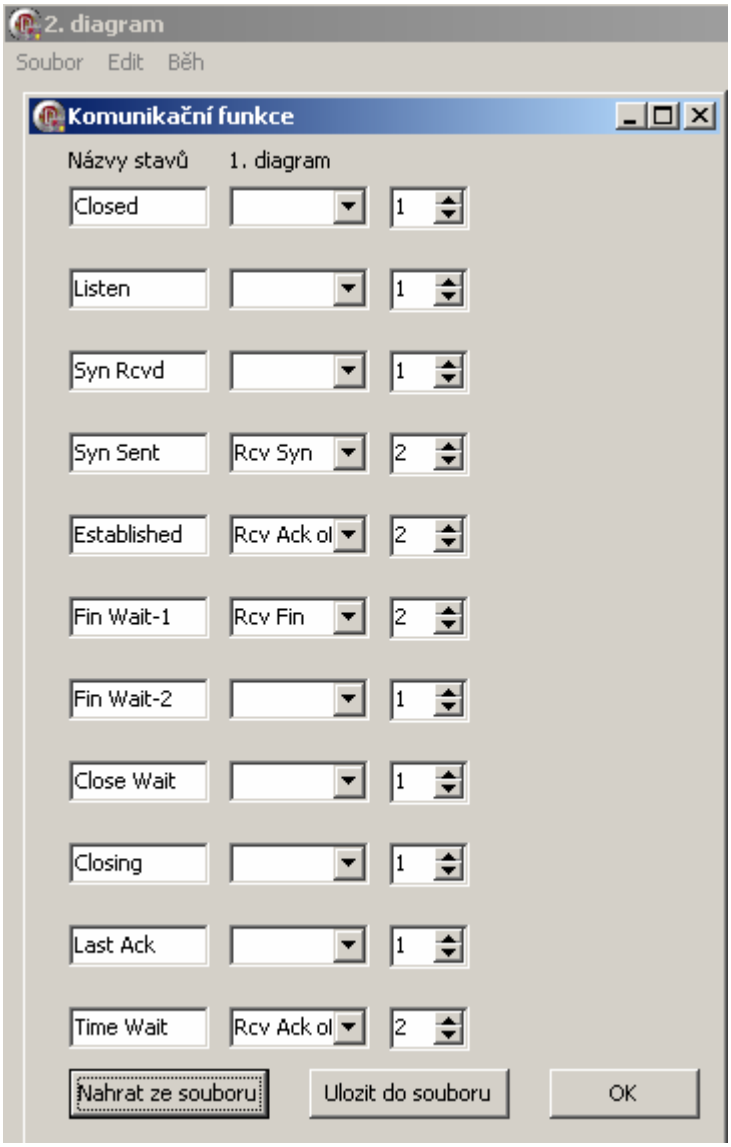
Komunikační funkce 1. diagramu vypadá takto:



Obr. 23 - Komunikační funkce A

Po příchodu do stavu Syn Rcvd, je vyslána událost Rcv Syn-Ack, stav Close Wait vyšle Rcv Ack of Fin a stav Last Ack vyšle Rcv Fin. Všem vysílaným událostem jsem přiřadil časový interval 2.

Komunikační funkce 2. diagramu vypadá takto:



Obr. 24 - Komunikační funkce B

Stavu Syn Sent vyvolá událost Rcv Syn, stav Established vyšle Rcv Ack of Syn, stav Fin Wait-1 vyšle Rcv Fin a stav Time Wait vyvolá Rcv Ack of Fin.

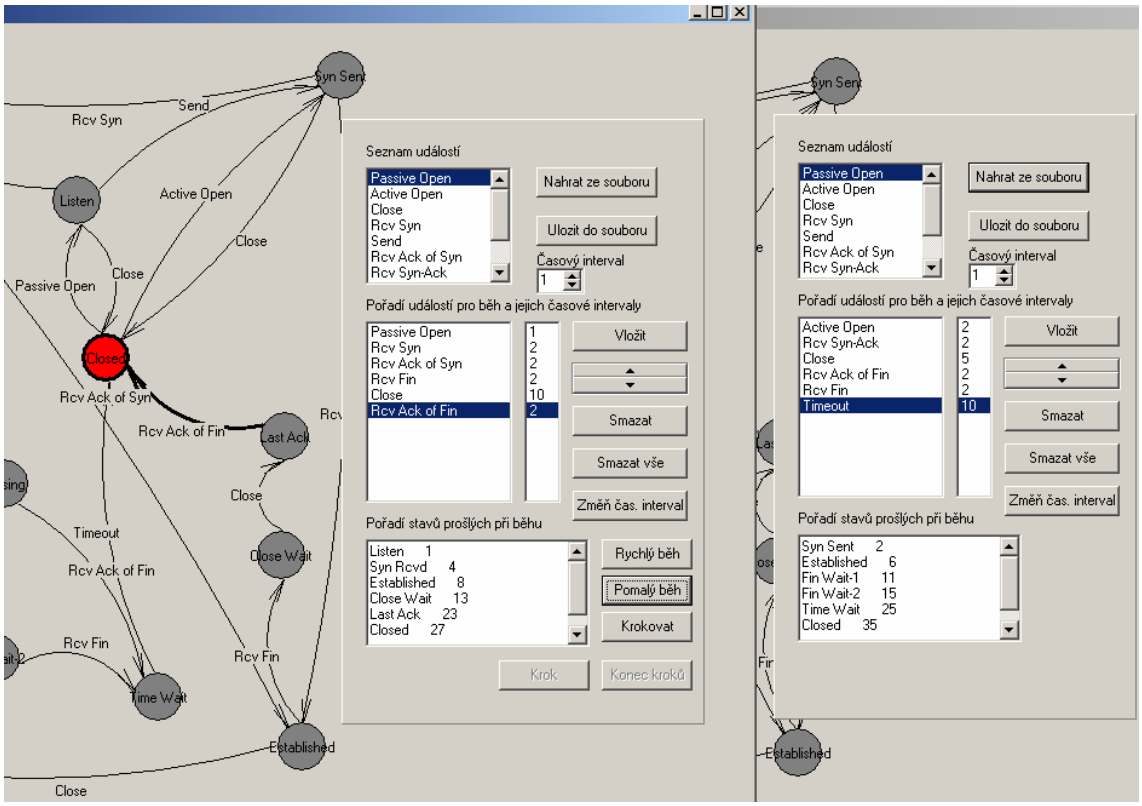
Simulace představuje situaci, kdy jedna strana požádá o spojení, druhá strana to akceptuje a ke spojení tudíž dojde. První strana odvysílá co potřebuje a vyšle žádost o ukončení spojení, druhá strana ještě chvíli vysílá a pak souhlasí s ukončením.

Průběh simulace jaký by měl být je znázorněn v tabulce.

Čas	Stav 1. diagramu	Vyslaná událost	Stav 2. digramu
0	Closed		Closed
1	Listen		
2		<= Rcv Syn	Syn Sent
4	Syn Rcvd	Rcv Syn-Ack =>	
6		<= Rcv Ack of Syn	Established
8	Established		
11		<= Rcv Fin	Fin Wait-1
13	Close Wait	Rcv Ack of Fin =>	
17			Fin Wait-2
23	Last Ack	Rcv Fin =>	
25		<= Rcv Ack of Fin	Time Wait
27	Closed		
35			Closed

Obr. 25 – Průběh čtvrté simulace

Výsledek simulace je vidět na obrázku:



Obr. 26 - Čtvrtá provedená simulace

Prošlé stavy i časové intervaly odpovídají předpokladům. V pořadí událostí je možné vidět jak se události zařadily a rozšířily tak původní seznamy. Komunikační funkce tak plní svou funkci.

Při všech simulacích se prokázalo, že aplikace pracuje správně i pro reálné stavové diagramy. Komunikace mezi více diagramy funguje také bezproblémově.

Závěr

Pomocí odborné literatury jsem se blíže seznámil s teorií stavových diagramů. Tyto poznatky jsem posléze aplikoval při tvorbě editoru a simulátoru. Podařilo se mi vytvořit prostředek pro snadné vytváření a simulování stavových diagramů a to i více najednou. Aplikace byla vyzkoušena na konkrétních stavových diagramech. Všechny diagramy byly úspěšně vytvořeny i nasimulovány. Výsledky simulací odpovídají teoretickým předpokladům. Pro potřeby aplikace jsem navrhl čtyři datové formáty, ve kterých jsou ukládány informace o součástech stavového diagramu. Byly tedy splněny všechny úkoly vyplývající ze zadání.

Seznam použité literatury

[1] H. Kanisová, M. Müller: UML srozumitelně, Computer Press, 2004. ISBN 80-251-0231-9

[2] M. Page-Jones: Základy objektově orientovaného návrhu v UML, Grada Publishing, 2001. ISBN 80-247-0210-X

[3] J. Schmuller: Myslíme v jazyku UML, Grada Publishing, 2001. ISBN 80-247-0029-8

[4] D Häring: Linuxový TCP stack, odolnost proti DoS a výkon(1), www.linuxzone.cz, 2005. Internetový naučný článek. URL: <http://www.linuxzone.cz/index.phtml?ids=4&idc=1367>

Příloha A – Deklarace třídy TState1

```
unit State1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, StdCtrls;

type
 TEventPath = record
    Event: String;
    Target: Integer;
 end;

type
TStatus=(sNone,sIO,sIN,sOUT);


type
  TState1 = class(TShape)
  private
    { Private declarations }
    FStatus: TStatus;
    FEventPath: Array of TEventPath;
    FTitle: String;
    function GetEventPath(const AIndex: Integer): TEventPath;
    procedure SetEventPath(const AIndex: Integer; EditEventPath: TEventPath);
  protected
    { Protected declarations }
  public
    { Public declarations }
    constructor Create(AOwner: TComponent); override;
    property EventPath[const AIndex: Integer]: TEventPath read GetEventPath write
SetEventPath;
    function GetL: Integer;
    procedure SetL(LIndex: Integer);
    procedure CreateTitle(Lab1: TLabel);
    procedure AlignTitle(Lab1: TLabel);
    procedure SetTitle(Lab1: TLabel; Tit: string);
```

```
published
   { Published declarations }
   property Status: TStatus read FStatus write FStatus;
   property Title: String read FTitle write FTitle;
   property PopupMenu;
  end;

implementation

constructor TState1.Create(AOwner: TComponent);
begin
      inherited Create(AOwner);
end;

function TState1.GetEventPath(const AIndex: Integer): TEventPath;
begin
      Result:=FEventPath[AIndex]
end;

procedure TState1.SetEventPath(const AIndex: Integer; EditEventPath: TEventPath);
begin
      FEventPath[AIndex]:=EditEventPath;
end;

function TState1.GetL: Integer;
begin
      Result:=Length(FEventPath)
end;

procedure TState1.SetL(LIndex: Integer);
begin
      SetLength(FEventPath,LIndex)
end;

procedure TState1.CreateTitle(Lab1: TLabel);
begin
      with Lab1 do
      begin
        Parent:=self.Parent;
        Top:=self.Top+14;
        Caption:=self.Title;
```

```
        Left:=self.Left+20-round(Width/2);
        Transparent:=true;
      end;
end;

procedure TState1.AlignTitle(Lab1: TLabel);
begin
      Lab1.Top:=self.Top+14;
      Lab1.Left:=self.Left+20-round(Lab1.Width/2);
end;

procedure TState1.SetTitle(Lab1: TLabel; Tit: string);
begin
      Lab1.Caption:=Tit;
      Lab1.Top:=self.Top+14;
      Lab1.Left:=self.Left+20-round(Lab1.Width/2);
end;
```

Příloha B – Deklarace procedury ConnectAllStates

```
procedure TForm1.ConnectAllStates;
var k,l,o,stred,dS,dE: integer;
   UsedPath: array of integer;
begin
   if State<>nil then
   begin
    PForm1.Image1.Canvas.FillRect(PForm1.GetClientRect);
    if Paths<> nil then
    begin
     dS:=Length(Paths)-1;
     dE:=Length(Paths[1])-1;
     for k:=0 to dS do
      for l:=0 to dE do
       If Paths[k,l]<> nil then
       begin
        Paths[k,l].Free;
        Paths[k,l]:=nil;
       end;
    end;
    SetLength(UsedPath,EC);
    SetLength(Paths,SC,EC);
    for k:=0 to SC-1 do
    begin
     for l:=0 to EC-1 do
      UsedPath[l]:=-1;
     State[k].BringToFront;
     Titles[k].BringToFront;
     for l:=0 to EC-1 do
      if k<>State[k].EventPath[l].Target then
      begin
       for o:=0 to EC-1 do
       if UsedPath[o]=State[k].EventPath[l].Target then
        break;
       if o=EC then
       begin
        Paths[k,l]:=TLabel.Create(PForm1);
        Paths[k,l].Parent:=PForm1;
        Paths[k,l].Caption:=State[k].EventPath[l].Event;
        ConnectStates(State[k],State[State[k].EventPath[l].Target],Paths[k,l]);
        UsedPath[l]:=State[k].EventPath[l].Target
```

```
          end else begin
            stred:=Paths[k,o].Left+Paths[k,o].Width div 2;
            Paths[k,o].Caption:=Paths[k,o].Caption+', '+State[k].EventPath[l].Event;
            Paths[k,o].Left:=stred-Paths[k,o].Width div 2;
          end;
        end;
      end;
    end;
end;
```

Příloha C – Deklarace procedury ConnectStates

```
procedure TForm1.ConnectStates(S1: TState1; S2: TState1; PathTit: TLabel);
var Bod1,Bod2,Bod3,Bod4,Stred,Sip1,Sip2: TPoint;
   Uhel,A,A2,Uhsip: Real;
   d,r,deltaT,deltaL,deltaY,deltaX: integer;
begin
     deltaT:=S2.Top-S1.Top;
     deltaL:=S2.Left-S1.Left;
     if deltaL <> 0 then
       Uhel:=ArcTan(deltaT/deltaL)
     else
       Uhel:=pi/2;
     A2:=(S1.Top+20+Sign(deltaL)*20*System.Sin(uhel)-(S2.Top+20-Sign(deltaL)*
20*System.Sin(uhel)))*(S1.Top+20+Sign(deltaL)*20*System.Sin(uhel)-(S2.Top+20-
Sign(deltaL)*20*System.Sin(uhel)))+(S1.Left+20+Sign(deltaL)*20*cos(uhel)-
(S2.Left+20-Sign(deltaL)*20*cos(uhel)))*(S1.Left+20+Sign(deltaL)*20*cos(uhel)-
(S2.Left+20-Sign(deltaL)*20*cos(uhel)));
     A:=Sqrt(A2)/2;
     d:=20+Round(A2/80);
     r:=round(d/2);
     Stred.X:=S1.Left+20+Sign(deltaL)*round(20*cos(uhel)+A*cos(uhel)-(r-
20)*System.Sin(uhel));
    Stred.Y:=S1.Top+20+Sign(deltaL)*round(20*System.Sin(uhel)+A*
System.Sin(uhel)+(r-20)*cos(uhel));
     Bod1.X:=-r+Stred.X;
     Bod1.Y:=-r+Stred.Y;
     Bod2.X:=r+Stred.X;
     Bod2.Y:=r+Stred.Y;
     Bod3.X:=S2.Left+20-round(Sign(deltaL)*20*cos(uhel));
     Bod3.Y:=S2.Top+20-round(Sign(deltaL)*20*System.Sin(uhel));
     Bod4.X:=S1.Left+20+round(Sign(deltaL)*20*cos(uhel));
     Bod4.Y:=S1.Top+20+round(Sign(deltaL)*20*System.Sin(uhel));

PForm1.Image1.Canvas.Arc(Bod1.X,Bod1.Y,Bod2.X,Bod2.Y,Bod3.X,Bod3.Y,Bod4.X
,Bod4.Y);
     deltaY:=Stred.Y-Bod3.Y;
     deltaX:=Bod3.X-Stred.X;
     if deltaX <> 0 then
       UhSip:=ArcTan(deltaY/deltaX)
     else
       UhSip:=pi/2;
     Sip1.X:=Bod3.X-round(20*Sign(deltaX)*cos(pi/2-UhSip+pi/36));
     Sip1.Y:=Bod3.Y-round(20*Sign(deltaX)*System.Sin(pi/2-UhSip+pi/36));
```

```
    Sip2.X:=Bod3.X-round(20*Sign(deltaX)*cos(pi/2-UhSip-3*pi/36));
    Sip2.Y:=Bod3.Y-round(20*Sign(deltaX)*System.Sin(pi/2-UhSip-3*pi/36));
    PForm1.Image1.Canvas.MoveTo(Sip1.X,Sip1.Y);
    PForm1.Image1.Canvas.LineTo(Bod3.X,Bod3.Y);
    PForm1.Image1.Canvas.LineTo(Sip2.X,Sip2.Y);
    with PathTit do
    begin
      Left:=Stred.X+Sign(deltaL)*round((r+10)*System.Sin(uhel))-Width div 2;
      Top:=Stred.Y-Sign(deltaL)*round((r+10)*cos(uhel))-Height div 2;
    end;
end;
```

Příloha D – Deklarace procedury ShowPath

```
procedure TForm1.ShowPath(S1: TState1; S2: TState1; Part: Integer);
var Bod1,Bod2,Bod3,Bod4,Bod5,Bod6,Stred,Sip1,Sip2: TPoint;
   Uhel,A,A2,Uhsip: Real;
   d,r,deltaT,deltaL,deltaY,deltaX: integer;
begin
    PForm1.Image1.Canvas.Pen.Width:=3;
    deltaT:=S2.Top-S1.Top;
    deltaL:=S2.Left-S1.Left;
    if deltaL <> 0 then
     Uhel:=ArcTan(deltaT/deltaL)
    else
     Uhel:=pi/2;
    A2:=(S1.Top+20+Sign(deltaL)*20*System.Sin(uhel)-(S2.Top+20-
Sign(deltaL)*20*System.Sin(uhel)))*(S1.Top+20+Sign(deltaL)*20*System.Sin(uhel)-
(S2.Top+20-
Sign(deltaL)*20*System.Sin(uhel)))+(S1.Left+20+Sign(deltaL)*20*cos(uhel)-
(S2.Left+20-Sign(deltaL)*20*cos(uhel)))*(S1.Left+20+Sign(deltaL)*20*cos(uhel)-
(S2.Left+20-Sign(deltaL)*20*cos(uhel)));
    A:=Sqrt(A2)/2;
    d:=20+Round(A2/80);
    r:=round(d/2);
    Stred.X:=S1.Left+20+Sign(deltaL)*round(20*cos(uhel)+A*cos(uhel)-(r-
20)*System.Sin(uhel));

Stred.Y:=S1.Top+20+Sign(deltaL)*round(20*System.Sin(uhel)+A*System.Sin(uhel)+(
r-20)*cos(uhel));
    Bod1.X:=-r+Stred.X;
    Bod1.Y:=-r+Stred.Y;
    Bod2.X:=r+Stred.X;
    Bod2.Y:=r+Stred.Y;
    Bod3.X:=S2.Left+20-round(Sign(deltaL)*20*cos(uhel));
    Bod3.Y:=S2.Top+20-round(Sign(deltaL)*20*System.Sin(uhel));
    Bod4.X:=S1.Left+20+round(Sign(deltaL)*20*cos(uhel));
    Bod4.Y:=S1.Top+20+round(Sign(deltaL)*20*System.Sin(uhel));
    If Part > 0 then
    begin
     Bod6.X:=Bod4.X+Round((Part-1)*(Bod3.X-Bod4.X)/3);
     Bod6.Y:=Bod4.Y+Round((Part-1)*(Bod3.Y-Bod4.Y)/3);
     Bod5.X:=Bod4.X+Round(Part*(Bod3.X-Bod4.X)/3);
     Bod5.Y:=Bod4.Y+Round(Part*(Bod3.Y-Bod4.Y)/3);
```

```
PForm1.Image1.Canvas.Arc(Bod1.X,Bod1.Y,Bod2.X,Bod2.Y,Bod5.X,Bod5.Y,Bod6.X
,Bod6.Y);
    end else

PForm1.Image1.Canvas.Arc(Bod1.X,Bod1.Y,Bod2.X,Bod2.Y,Bod3.X,Bod3.Y,Bod4.X
,Bod4.Y);
    if (Part = 3) or (Part = 0) then
    begin
     deltaY:=Stred.Y-Bod3.Y;
     deltaX:=Bod3.X-Stred.X;
     if deltaX <> 0 then
       UhSip:=ArcTan(deltaY/deltaX)
     else
       UhSip:=pi/2;
     Sip1.X:=Bod3.X-round(20*Sign(deltaX)*cos(pi/2-UhSip+pi/36));
     Sip1.Y:=Bod3.Y-round(20*Sign(deltaX)*System.Sin(pi/2-UhSip+pi/36));
     Sip2.X:=Bod3.X-round(20*Sign(deltaX)*cos(pi/2-UhSip-3*pi/36));
     Sip2.Y:=Bod3.Y-round(20*Sign(deltaX)*System.Sin(pi/2-UhSip-3*pi/36));
     PForm1.Image1.Canvas.MoveTo(Sip1.X,Sip1.Y);
     PForm1.Image1.Canvas.LineTo(Bod3.X,Bod3.Y);
     PForm1.Image1.Canvas.LineTo(Sip2.X,Sip2.Y);
    end;
    PForm1.Image1.Canvas.Pen.Width:=1;
end;
```

Příloha E – Deklarace funkce NextState

```
function TForm1.NextState(ActualEvent: String; ActualState: integer): integer;
var l:  integer;
begin
     for l:=0 to EC-1 do
       if State[ActualState].EventPath[l].Event=ActualEvent then
         break;
       Result:=State[ActualState].EventPath[l].Target;
end;
```

Příloha F – Deklarace procedury ComSD

```
procedure TForm1.ComSD(IndSt: Integer);
var k,pomi:integer;
begin
    if PForm1=Form1 then
    begin
     for k:=0 to DC-1 do
      if MemoryIE[IndSt,k]>0 then
      begin
       if Diags[k].Kon then
       begin
         Diags[k].EventOrder.Items.Insert(
Diags[k].EventOrder.ItemIndex+1,Diags[k].MemoryEL[MemoryIE[IndSt,k]-1]);
         Diags[k].TimeInt.Items.Insert(
Diags[k].TimeInt.ItemIndex+1,IntToStr(MemoryITI[IndSt,k]));
         Diags[k].Kon:=false;
        end else begin
         Diags[k].EventOrder.Items.Insert(
Diags[k].EventOrder.ItemIndex,Diags[k].MemoryEL[MemoryIE[IndSt,k]-1]);
         Diags[k].TimeInt.Items.Insert(
Diags[k].TimeInt.ItemIndex,IntToStr(MemoryITI[IndSt,k]));
         Diags[k].EventOrder.ItemIndex:=Diags[k].EventOrder.ItemIndex-1;
         Diags[k].TimeInt.ItemIndex:=Diags[k].EventOrder.ItemIndex;
        end;
        Diags[k].TimeValue:=TimeValue;
      end;
    end else begin
     for k:=1 to DC-1 do
     begin
      if k>=Poradi-1 then
       pomi:=k
      else
       pomi:=k-1;
      if MemoryIE[IndSt,pomi]>0 then
      begin
       if Diags[pomi].Kon then
       begin

Diags[pomi].EventOrder.Items.Insert(Diags[pomi].EventOrder.ItemIndex+1,Diags[pom
i].MemoryEL[MemoryIE[IndSt,pomi]-1]);

Diags[pomi].TimeInt.Items.Insert(Diags[pomi].TimeInt.ItemIndex+1,IntToStr(Memory
ITI[IndSt,pomi]));
```

```
          Diags[pomi].EventOrder.ItemIndex:=Diags[pomi].EventOrder.ItemIndex+1;
          Diags[pomi].TimeInt.ItemIndex:=Diags[pomi].EventOrder.ItemIndex;
          Diags[pomi].Kon:=false;
        end else begin

Diags[pomi].EventOrder.Items.Insert(Diags[pomi].EventOrder.ItemIndex,Diags[pomi].
MemoryEL[MemoryIE[IndSt,pomi]-1]);

Diags[pomi].TimeInt.Items.Insert(Diags[pomi].TimeInt.ItemIndex,IntToStr(MemoryIT
I[IndSt,pomi]));
          Diags[pomi].EventOrder.ItemIndex:=Diags[pomi].EventOrder.ItemIndex-1;
          Diags[pomi].TimeInt.ItemIndex:=Diags[pomi].EventOrder.ItemIndex;
        end;
        Diags[pomi].TimeValue:=TimeValue;
      end;
    end;
    if MemoryIE[IndSt,0]>0 then
    begin
      if Form1.Kon then
      begin

Form1.EventOrder.Items.Insert(Form1.EventOrder.ItemIndex+1,Form1.MemoryEL[Me
moryIE[IndSt,0]-1]);

Form1.TimeInt.Items.Insert(Form1.TimeInt.ItemIndex+1,IntToStr(MemoryITI[IndSt,0]
));
        Form1.EventOrder.ItemIndex:=Form1.EventOrder.ItemIndex+1;
        Form1.TimeInt.ItemIndex:=Form1.EventOrder.ItemIndex;
        Form1.Kon:=false;
      end else begin

Form1.EventOrder.Items.Insert(Form1.EventOrder.ItemIndex,Form1.MemoryEL[Mem
oryIE[IndSt,0]-1]);

Form1.TimeInt.Items.Insert(Form1.TimeInt.ItemIndex,IntToStr(MemoryITI[IndSt,0]));
        Form1.EventOrder.ItemIndex:=Form1.EventOrder.ItemIndex-1;
        Form1.TimeInt.ItemIndex:=Form1.EventOrder.ItemIndex;
      end;
      Form1.TimeValue:=TimeValue;
    end;
  end;
end;
```

Příloha G – Deklarace procedury NextStep

```
procedure TForm1.NextStep(AParam: Integer);
var NewState:integer;
begin
    NewState:=NextState(EventOrder.Items[EventOrder.ItemIndex],ActState);
    if ActState<>NewState then
    begin
     State[ActState].Brush.Color:=clGray;
     State[NewState].Brush.Color:=clRed;
     if AParam=1 then
     begin
       ConnectAllStates;
       ShowPath(State[ActState],State[NewState],0);
     end;
    end;
    if KonecBhu2.Enabled then
     UsedStates.Lines.Append(StateNames[NewState]+'      '+IntToStr(TimInts[Poradi-
1]))
    else
     UsedStates.Lines.Append(StateNames[NewState]);
    ActState:=NewState;
    if EventOrder.ItemIndex<EventOrder.Items.Count-1 then
    begin
     EventOrder.ItemIndex:=EventOrder.ItemIndex+1;
     TimeInt.ItemIndex:=EventOrder.ItemIndex;
    end else
     Kon:=true;
    if KonecBhu2.Enabled then
     ComSD(ActState);
end;
```

Příloha H – Deklarace procedury NextStepOT

```
procedure TForm1.NextStepOT(TC: Integer);
var NewState:integer;
begin
    PForm1.Show;
    NewState:=NextState(EventOrder.Items[EventOrder.ItemIndex],ActState);
    if ActState<>NewState then
    begin
     if TC= 1 then
      ConnectAllStates;
     if TC= 2 then
      State[ActState].Brush.Color:=clGray;
     if TC= 3 then
      State[NewState].Brush.Color:=clRed;
     ShowPath(State[ActState],State[NewState],TC);

     if TC=3 then
     begin
      if KonecBhu2.Enabled then
       UsedStates.Lines.Append(StateNames[NewState]+'
'+IntToStr(TimInts[Poradi-1]))
      else
       UsedStates.Lines.Append(StateNames[NewState]);
      ActState:=NewState;
      if EventOrder.ItemIndex<EventOrder.Items.Count-1 then
      begin
       EventOrder.ItemIndex:= EventOrder.ItemIndex+1;
       TimeInt.ItemIndex:=EventOrder.ItemIndex;
      end else
       Kon:=true;
      if KonecBhu2.Enabled then
      begin
       TimeValue:=TimInts[Poradi-1];
       ComSD(ActState);
      end;
     end;
    end else begin
     if EventOrder.ItemIndex<EventOrder.Items.Count-1 then
      begin
       EventOrder.ItemIndex:= EventOrder.ItemIndex+1;
       TimeInt.ItemIndex:=EventOrder.ItemIndex;
      end else
```

```
          Kon:=true;
       UsedStates.Lines.Append(StateNames[NewState]+'       '+IntToStr(TimInts[Poradi-
1]));
       TimeValue:=TimInts[Poradi-1];
       ComSD(ActState);
       Form1.TickCount:=0;
     end;
end;
```