VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

VÝUKOVÝ SIMULÁTOR V PROSTŘEDÍ WEBASSEMBLY

EDUCATIONAL SIMULATOR IN WEBASSEMBLY ENVIRONMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **JAROMÍR BŘEZINA**
AUTHOR

VEDOUCÍ PRÁCE — **Dr. Ing. PETR PERINGER**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Březina Jaromír**
Program: Informační technologie
Název: **Výukový simulátor v prostředí WebAssembly**
Educational Simulator in WebAssembly Environment
Kategorie: Modelování a simulace

Zadání:

1. Seznamte se s existujícími implementacemi simulátorů blokových schemat a Petriho sítí. Seznamte se s vytvářením aplikací v prostředí *WebAssembly*.
2. Navrhněte simulátor a jednoduchý editor blokových schemat a časovaných stochastických Petriho sítí. Navrhněte intuitivní grafické uživatelské rozhraní s modulem pro vizualizaci výsledků simulace. Navrhněte demonstrační příklady vhodné pro výuku v předmětu IMS.
3. Navržený simulátor implementujte v C++ s využitím prostředí *WebAssembly* a vhodně zvolených knihoven. Implementujte sadu alespoň 10 příkladů vhodných pro výuku.
4. Aplikaci řádně otestujte, zhodnoťte dosažené výsledky a navrhněte další možnosti vývoje této aplikace.

Literatura:

- Domovská stránka projektu *WebAssembly* (https://webassembly.org/).
- Studijní materiály pro předmět IMS.
- Další dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Peringer Petr, Dr. Ing.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 3. listopadu 2021

Abstrakt

Tato bakalářská práce se zaměřuje na problematiku simulace časovaných stochastických Petriho sítí a spojitých blokových schémat pro výukové účely. Výsledná aplikace je tvořena ze dvou dílčích částí. První z nich je jednoduchý grafický editor, jenž poskytuje nástroje pro vytvoření požadovaného modelu ve formě Petriho sítě nebo blokového schématu a vizualizaci výsledků simulace. Grafické uživatelské rozhraní editoru bylo vytvořeno pomocí knihovny React. Druhou část aplikace tvoří simulátor implementovaný v jazyce C++, jenž slouží pro simulaci vytvořeného modelu a je integrován do celého řešení jako WebAssembly modul. Součástí aplikace je i sada 10 příkladů, které slouží jako ukázka jednoduchých modelů vhodných pro výukové účely.

.

Abstract

This bachelor thesis focuses on the issue of simulation of timed stochastic Petri nets and continuous block diagrams for educational purposes. The resulting application consists of two sub-parts. The first is a simple graphical editor that provides tools to create the desired model in the form of a Petri net or block diagram and visualize the simulation results. The graphical user interface of the editor was created using the React library. The second part of the application consists of a simulator implemented in C++, which is used to simulate the created model and is integrated into the entire solution as a WebAssembly module. The application also includes a set of 10 examples that serve as a demonstration of simple models suitable for educational purposes.

Klíčová slova

Simulace, algoritmus řízení simulace, Petriho sítě, Webassembly, spojitá bloková schémata, numerické metody, editor, C++, React, Redux, Typescript

Keywords

Simulation, algorithm controlled simulation, Petri nets, WebAssembly, continuous block schemes, numerical methods, editor, C++, React, Redux, Typescript

Citace

BŘEZINA, Jaromír. *Výukový simulátor v prostředí WebAssembly.* Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Dr. Ing. Petr Peringer

Výukový simulátor v prostředí WebAssembly

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doktora Petra Peringera. Uvedl jsem všechny bibliografické a elektronické zdroje, ze kterých jsem čerpal.

.....
Jaromír Březina
10. května 2022

Poděkování

Mé poděkování patří Dr. Ing. Petru Peringerovi za cenné rady, odborné vedení, věcné připomínky, trpělivost a vstřícnost při konzultacích této práce.

Obsah

1 Úvod **2**

2 Vysvětlení pojmů a současný stav **3**
2.1 WebAssembly 3
2.2 Emscripten 5
2.3 Petriho sítě 6
2.4 Blokové schéma 8
2.5 Přehled současných řešení 9
2.6 Použité technologie 13

3 Specifikace a návrh řešení **17**
3.1 Specifikace výsledné aplikace 17
3.2 Volba způsobu realizace 18
3.3 Návrh simulátoru 18
3.4 Návrh editoru 20

4 Implementace **22**
4.1 Založení projektu 22
4.2 Implementace editoru 23
4.3 Implementace simulátoru 24
4.4 Dostupnost 27
4.5 Testování 27

5 Příklady modelů **28**
5.1 Příklady pro Petriho Sítě 28
5.2 Příklady pro spojitá bloková schéma 31

6 Závěr **34**

Literatura **35**

Kapitola 1

Úvod

Cílem této bakalářské práce je implementovat jednoduchý editor a simulátor časovaných stochastických Petriho sítí a spojitých blokových schémat. Současně je cílem práce také demonstrovat využití technologie WebAssembly při vývoji webových aplikací. Použití výsledné aplikace je směřováno pro účely výuky zmíněných dvou oblastí a pro tento účel byla implementována sada demonstračních příkladů. Důraz byl kladen na maximální dostupnost a snadné použití. Komplexních nástrojů pro modelování a simulaci je na trhu dostupná celá řada, kdy část z nich je analyzována v první kapitole, avšak ty kvůli své složitosti mohou většinu studentů spíše odradit.

Druhá kapitola je věnována technologii WebAssembly, vysvětlení problematiky Petriho Sítě, včetně dalších odvozených variant a blokových schémat. Závěr kapitoly je věnován jazykům, knihovnám a dalším nástrojům, jenž byly použity pro vyhotovení aplikace. V pořadí třetí kapitola obsahuje specifikaci a návrh výsledné aplikace. Je možné se v ní dovědět, co vedlo ke zvolenému způsobu realizace a výběru technologií. Dále obsahuje detailnější specifikaci požadované funkčnosti, návrhy jednotlivých částí a strukturu grafického uživatelského rozhraní. Následující čtvrtá kapitola je určena procesu implementace a obsahuje informace k důležitým algoritmům, třídám a strukturám. Pro editor se kapitola věnuje vybraným komponentám, jenž tvoří prezentační vrstvu. Pro část simulátoru pak kapitola obsahuje použité algoritmy pro simulaci. Dále se zde nachází, jak byl realizován překlad simulátoru do WebAssembly modulu a jeho propojení pomocí adaptéru s editorem. Předposlední pátá kapitola se věnuje vytvořené sadě příkladů, konkrétně jaké příklady jsou v aplikaci obsaženy pro jednotlivé oblasti a co mají demonstrovat. V závěru jsou pak zhodnoceny dosažené výsledky a navržen budoucí vývoj.

Kapitola 2

Vysvětlení pojmů a současný stav

V následující kapitole se čtenář nejprve dozví, co je to WebAssembly, jak funguje a k čemu je možné tuto technologii použít. Posléze je čtenář seznámen s tím, co jsou to Petriho sítě a blokové schéma. Po této sekci následuje rozbor stávajících řešení pro modelování a simulace Petriho sítí nebo blokových schémat. Sekce v závěru kapitoly se věnuje jazykům, knihovnám a technologiím, jenž budou použity pro vyhotovení výsledné aplikace.

2.1 WebAssembly

WebAssembly[22], přičemž je možné používat zkratku *Wasm*, specifikuje formát zápisu binárních instrukcí spolu s odpovídajícím jazykem. Tento jazyk lze přirovnat k jazyku symbolických adres a kód zapsaný v tomto standardu je spustitelný ve webovém prohlížeči, který slouží jako hostující prostředí. Odpovídající pseudo-jazyk není určený k zapisování programu přímou cestou, ale spíše jako cíl kompilace pro jazyky vyšší úrovně. Těmito jazyky mohou být například C, C++, C# nebo Rust.

Poprvé byl tento standard oznámen v roce 2015 a vydání první verze se konalo o dva roky později. O udržování této technologie se stará konsoricum *W3C*, avšak za vývojem stojí společnosti Mozilla, Microsoft, Google a Apple. Díky tomu je podpora pro WebAssembly integrována ve všech prohlížečích, které tyto společnosti nabízejí, tedy Firefox, Edge, Chrome i Safari.

Webassembly specifikuje dvě přípony názvu souboru, a to .wat a .wasm. První z nich je textový formát zápisu instrukcí a druhá přípona pro výsledný binární formát. Pro kompilaci .wat souborů lze použít překladač wat2wasm. Jak oba formáty zápisu vypadají je možné vidět na obrázku 2.1

Pro použití funkcí z wasm souboru je potřeba ho v době běhu webové aplikace načíst, nechat prohlížeč zkompilovat soubor do WebAssembly modulu a pro tento modul následně vytvořit instanci. JavaScript poskytuje API pro práci s wasm soubory a pro realizaci jednotlivých kroků. Z objektu modulu je pak možné volat požadované metody a zdroje.

Vlastnosti a případy užití

Primární kladnou vlastností je, že rychlost provedení kódu zapsaného ve WebAssembly je srovnatelná s rychlostí nativního strojového kódu. Účelem této technologie není kompletní nahrazení jazyku JavaScript při vývoji webových aplikací, naopak WebAssembly a JavaScript mají mezi sebou koexistovat a vzájemně se doplňovat v oblastech, ve kterých zaostávají.

Obrázek 2.1: Ukázka WAT a WASM formátu.

Jedním z případů použití WebAssembly je jako prostředek pro zrychlení částí webové aplikace, které jsou výpočetně náročné. Největší využití se nabízí pro hry, zpracování obrazu a zvuku, šifrování nebo právě pro simulace. Tuto technologii lze také využít pro import stávajících knihoven nebo projektů napsaných pouze pro desktop do webového prostředí.

Použití WASM a JS pro tvorbu aplikací

Při využití WebAssembly v aplikaci se uvažují následující tři scénáře, jenž byly převzaty z [25]:

- Realizovat celou aplikaci v požadovaném jazyce včetně grafického uživatelského rozhraní a následně tuto aplikaci přeložit WebAssembly. Uživatelské rozhraní je v tomto případě vykreslováno do HTML5 canvas elementu využívající standard WebGL. Při tomto přístupu je JavaScript použit pouze pro načtení modulu s aplikací, vytvoření instance modulu a propojení s odpovídajícím elementem. Zmíněný přístup se používá zejména u her.
- Kombinace předchozího přístupu, kdy canvas element řízený z WebAssembly tvoří hlavní okno, kolem kterého jsou umístěny další UI prvky vytvořené pomocí JS/HTML/CSS. WebAssembly modul obsahuje v tomto případě veškerou řídící logiku aplikace. Tento postup byl použit například u webové verze programu CAD[1].
- Posledním přístupem je celou aplikaci vytvořit pomocí JS/HTML/CSS společně s několika WebAssembly moduly, jenž slouží jako knihovny obsahující algoritmy pro výpočetně náročné operace.

Budoucí vývoj

Technologie WebAssembly je stále ve vývoji. Na GitHub stránkách projektu[2] je dostupný seznam, který obsahuje návrhy pro budoucí rozvoj. Mezi rozvoji se nachází například[15]:

- *Exception Handling* - V současné chvíli dojde při vyvolání výjimky pouze k přerušení vykonávání kódu. Po začlenění tohoto rozšíření by mělo být možné předávat řízení obsluhy výjimky do hostitelského prostředí, tedy předat řízení zpět do JavaScriptu.

[1] https://web.autocad.com/acad/me
[2] https://github.com/WebAssembly

- *Garbage Collector* - Cílem tohoto rozšíření je, aby *Garbage Collector* hostitelského prostředí měl možnost spravovat i objekty vytvořené ve WebAssembly modulu.
- *Threads* - Umožní využívání hostitelských prostředků pro paralelismus.
- *Reference Types* - Umožní předávání referencí na objekty z hostitelského prostředí. Jinými slovy umožní předání JavaScript objektů do Wasm modulu.

Wasmtime

Protože se technologie WebAssembly mezi vývojáři osvědčila jako prostředek pro tvorbu rychlých, bezpečných a škálovatelných aplikací, které lze spustit na různých strojích, kde je dostupný webový prohlížeč, vznikla myšlenka vytvoření samostatného běhového prostředí, které by bylo nezávislé na prohlížeči. Právě pro tento účel vzniklo systémové rozhraní *WebAssembly System Interface*[11], zkráceně *Wasi* a běhové prostředí *Wasmtime*[7].

Jazyk *WebAssembly* byl navržen tak, aby byl nezávislý na svém okolí, se kterým komunikuje výhradně prostřednictvím rozhraní. V případě běhu v prohlížeči je pro komunikaci využíváno *Web Api*. Rozhraní Wasi je v podstatě obdobou Web Api, které může být implementováno běhovým prostředím mimo prohlížeč. *Wasmtime* je pak běhové prostředí, které díky Wasi umožňuje interpretovat kód zapsaný ve WebAssembly na desktopu i webovém prohlížeči, kde je dostupné jako *polyfill*.

2.2 Emscripten

Emscripten [16] je volně dostupná sada nástrojů umožňující překlad zdrojových kódů do Webassembly. Před uvedením standardu WebAssembly byl primárním cílem kompilace jazyk Asm.js a sada stále obsahuje prostředky i pro tuto možnost. Nástroj je možné použít pro všechny programovací jazyky, které pro překlad využívají technologii LLVM. Primární zaměření je udáno pro jazyk C a C++. Projekt Emscripten je open-source projekt a je dostupný pod licencí MIT.

Emcc

Hlavním nástrojem sady jsou kompilátory *emcc*[16] pro jazyk C a *em++*[16] pro C++, které se starají o překlad zdrojového kódu do jazyku WebAseembly. Stejně jako u jiných kompilátorů i u těchto překladačů je možné samotný překlad modifikovat pomocí přepínačů. Při výchozím nastavení jsou výsledkem kompilace dva soubory. Prvním z nich s příponou .wasm obsahuje výsledný přeložený zdrojový kód ve WebAssembly. Druhým je JavaScript soubor s metodami pro runtime obsluhu prvního souboru. Vývojář pak pracuje primárně s tímto skriptem, který se postará o načtení, kompilaci i vytvoření instance modulu místo něj.

Emcmake

Další pomůckou, kterou sada obsahuje je program *emcmake*[16] pro práci s projekty využívajících CMake pro sestavení. Emcmake provede ve struktuře projektu potřebné změny, zajistí použití odpovídajícího kompilátoru pro překlad a předá požadované přepínače ovlivňující překlad.

Embind

Protože C/C++ a JavaScript nejsou typově identické jazyky, vzniká potřeba aparátu, který by tento problém vyřešil. Emscripten proto přišlo s knihovnou Embind[16], pomocí níž lze typy mezi oběma technologiemi provázat. Tato knihovna umožňuje vytvořit vazby pro funkce, třídy a další konstrukce jazyka tak, aby je vývojář mohl pohodlně použít při volání z WebAssembly. Nevýhodou však je, že vývojář musí pro všechny zdroje, které chce takto používat, definovat vazby ručně pomocí kódu. To může výt v případě velkých projektů značně zdlouhavý proces.

2.3 Petriho sítě

Petriho sítě, v roce 1962, vytvořil a popsal Carl Adam Petri ve své disertační práci [24]. Tyto sítě jsou popisovány jako nástroj, sloužící k modelování paralelních systémů a systémů s diskrétním časem. V průběhu let prošly Petriho sítě výrazným vývojem. Tento vývoj dal vzniknout celé řadě dalších variant, které původní definici určitým způsobem rozšiřují nebo jinak modifikují. V následující pasáži je uvedena formální definice, jak ji ve své práci uvádí Desel a Juhás [13].

Matematická definice

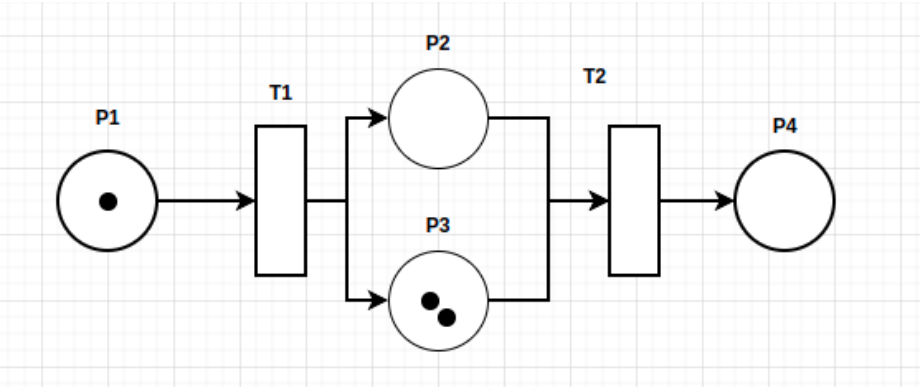
Petriho síť je pětice (S, T, F, M_0, W), kde:

- S je konečná množina míst.
- T je konečná množina přechodů.
- F je množina hran spojující přechody a místa.
- M_0 je počáteční značení.
- W je množina vah, které jsou přiřazeny každé hraně.

Popis struktury

Petriho síť je složena ze tří typů elementů, a to z *přechodů* (transition), *míst* (place), a orientovaných *hran* (arc). Hrany spojují pouze přechody a místa, nikdy nesmí spojovat elementy stejného typu. Nejčastěji se značí prostřednictvím šipky. Každé místo může obsahovat kladný počet tokenů, přičemž rozložení tokenů mezi místa se nazývá *značení*. Pro vyobrazení místa je využíváno označení pomocí prázdného kruhu, v němž se nachází černé tečky. Tyto tečky svým množstvím odpovídají počtu tokenů v daném místě. Pokud hrana spojuje místo s přechodem, označuje se jako *vstupní místo* pro daný přechod. Naopak v případě, kdy do místa hrana vede, je toto místo *výstupním místem* tohoto přechodu. Pokud všechna vstupní místa přechodu obsahují dostatečný počet tokenů, který je daný váhou hrany, dojde ke splnění podmínky pro provedení přechodu. Provedení přechodu se také nazývá "odpálení" přechodu a symbolizuje vznik nějaké události v systému. Jeho vykonáním dojde ke změně stavu modelu. Samotné "odpálení" se odehrává v nulovém čase a skládá se ze dvou kroků. V prvním kroku jsou odebrány tokeny ze vstupních míst podle váhy odpovídající hrany. Pomocí stejného kritéria jsou v druhém kroku tokeny umístěny do všech *výstupních míst* daného přechodu. Tento proces je opakován stále dokola, do doby dokud

lze vždy alespoň jeden přechod provést. Na obrázku 2.2 je možné vidět ukázku grafického zápisu Petriho sítě.



Obrázek 2.2: Příklad grafického zápisu Petriho sítě.

Další varianty Petriho sítí

Variant Petriho sítí existuje celá řada a text následující sekce se věnuje jen vybrané části z nich. Motivací pro vznik dalších variant bylo zvýšení modelovací síly. U jednotlivých variant jsou stručně rozebrány jejich hlavní rozdíly a charakteristiky.

Petriho sítě s inhibičními hranami

Tato varianta rozšiřuje původní definici o speciální typ vstupní hrany, která se nazývá inhibiční. Inhibiční hrana má pak opačnou sémantiku než normální hrana, tedy že podmínka je splněna v případě, že místo které hrana spojuje s přechodem žádný token neobsahuje. V grafech bývá inhibiční hrana značena pomocí linie zakončené vyplněným kolečkem.

Petriho sítě s prioritami přechodů

Každému přechodu je možné přiřadit hodnotu udávající jeho prioritu. V případě, že je možné provést více přechodů, tak přechody s vyšší prioritou jsou vykonány před přechody s prioritou nižší.

Časované a Stochastické časované Petriho sítě

Dvěma z celé řady možných rozšíření jsou i časované a stochastické časované Petriho sítě. Časovaná varianta rozšiřuje původní formalismus o pojem času a díky tomu lze tuto modifikaci lépe aplikovat na problémy reálného světa. V takové síti jsou pak rozlišeny dva typy přechodů, *okamžité* a *časované*. Okamžité přechody mohou být provedeny, pokud jsou splněny všechny jeho vstupní podmínky, zatímco u časovaného přechodu musí dojít i k uplynutí stanoveného času, nebo-li zpoždění. Zpoždění přechodu může být dáno deterministicky nebo může být dáno výsledkem stochastické funkce. V případě druhém pak mluvíme o stochastické časované Petriho síti.

Hierarchické Petriho sítě

Hlavní motivací pro vznik následující varianty byl problém v případě modelování komplexních a rozsáhlých systémů pomocí Petriho sítě, kdy se u takových modelů ztrácela přehlednost. Řešení pro tento problém poskytují Hierarchické Petriho sítě, jenž do standardní definice přidávají konstrukce pro lepší strukturování sítě do podsítí, které lze pak v modelu použít opakovaně.

Barvené Petriho sítě

Barvené Petriho sítě[21] kombinují klasické Petriho sítě s vlastnostmi programovacích jazyků vyšší úrovně. Hlavním rozdílem je, že tokeny zde mohou nést data, která jsou označovaná jako *barva tokenu*. Každému místu je pak přiřazena množina barev tokenů, jenž se mohou v daném místě nacházet. Tato množina v podstatě reprezentuje datový typ tokenů.

Další rozšíření je v podobě výrazů, které jsou součástí hran. V případě vstupní hrany pak výraz modifikuje proveditelnost tak, že aby byl daný přechod proveditelný, musí být mimo podmínky přítomnosti dostatečného počtu tokenů v místě, také splněn daný výraz hrany. Výrazy výstupních hran pak slouží k modifikování barev v nově vzniklých tokenech.

Přechod je možné rozšířit o tzv. *guard*, což je další výraz, jehož výsledkem je hodnota boolovského typu. *Guard* slouží pro přidání dalších omezujících podmínek určujících, zda-li je daný přechod proveditelný pro dané značení nebo ne.

2.4 Blokové schéma

Blokové schéma slouží pro grafický zápis modelu systému pomocí bloků. Základní stavební jednotkou je blok, který se skládá ze vstupního signálu, vnitřní funkce a výstupního signálu, přičemž vnitřní funkce slouží k přetransformování vstupního signálu na výstupní. Diagram bloku je v modelu realizován čtvercem.

Výstupy a vstupy jednotlivých bloků lze vzájemně propojit a tím dosáhnout k přenosu informací mezi nimi. Pro propojení bloků je použita čára nebo šipka. Postupným spojováním a kombinováním různých bloků lze modelovat komplexnější systémy nebo naopak dekomponovat systém na jeho pod-systémy. Jednou z výhod zápisu pomocí blokových schémat je přehlednost a na první pohled jasná struktura takto zapsaného systému.

Spojité blokové schéma

V praxi existuje mnoho kategorií blokových schémat a jejich uplatnění. Jednou z těchto kategorií je i spojité blokové schéma [23], jenž mohou sloužit pro modelování systému, který se skládá ze spojitých prvků. Chování takového systému je definováno pro každý časový okamžik.

Konkrétním příkladem aplikace na reálný problém je zápis diferenciálních rovnic pomocí funkčních a stavových bloků. Jednotlivé matematické operace obsažené v rovnici lze reprezentovat funkcí, která je následně přiřazena každému bloku a operandy jako vstupy a výstupy jednotlivých bloků.

Funkční bloky jsou bezpaměťové a slouží pro zapsání matematických funkcí a operací jako jsou sčítání, odčítání, násobení a tak podobně. Stavové bloky naopak uchovávají svůj stav v interní paměti a tento stav pak vstupuje do výpočtu. Zároveň vyžadují nastavení počátečních podmínek. Zástupcem stavových bloků je Integrátor. Tento typ bloku provádí

matematickou funkci integrování. Ke svému vnitřnímu fungování používá některou z numerických metod.

Numerické metody

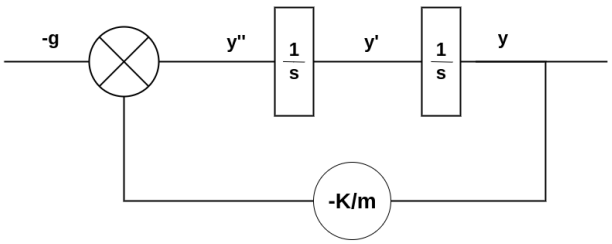
Numerické metody používáme pro řešení matematických úloh v případech, kdy je nalezení analytického řešení příliš složité nebo dokonce nemožné. Numerická metoda obsahuje popis jednotlivých kroků, jenž vedou k řešení úlohy. Nejjednodušší numerickou metodou je Eulerova metoda.

Eulerova metoda

Eulerovu metodu[23] publikoval švýcarský matematik a fyzik Leonhard Euler v roce 1768. Jedná se o jednokrokovou metodu pro řešení obyčejných diferenciálních rovnic s počáteční podmínkou. Eulerova metoda má následující tvar: $y(t+h) = y(t) + hf(t, y(t))$

Kde:

- h je délka kroku
- $y(t)$ je hodnota funkce v čase t
- $f(t, y(t))$ je derivací funkce y podle času t



Obrázek 2.3: Příklad blokového schématu.

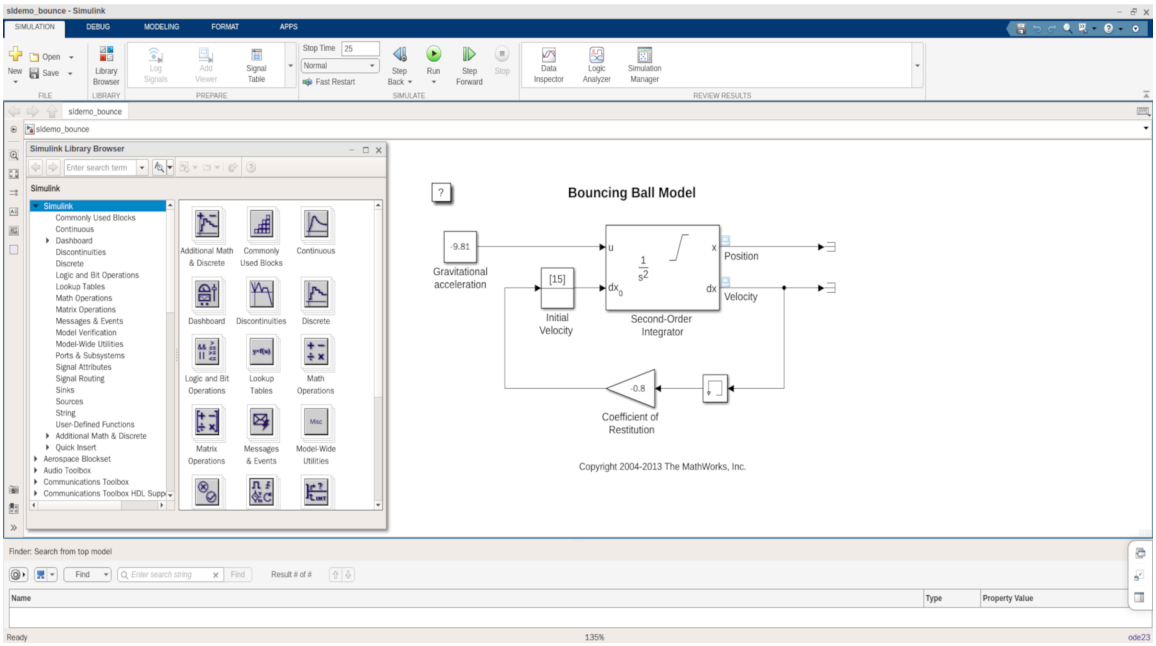
2.5 Přehled současných řešení

Simulink

Simulink[20] je komerční grafické prostředí nabízející široké spektrum profesionálních prostředků pro modelování a simulaci multi-doménových dynamických systémů. Jedná s o nadstavbu prostředí pro vědecko-technické výpočty MATLAB od stejné společnosti *The MathWorks*. Primárním nástrojem je grafický editor pro tvorbu modelů prostřednictvím blokových schémat. Pro uživatele je dostupná stabilní knihovna bloků pro celou řadu oblastí. Tuto knihovnu si navíc může uživatel libovolně přizpůsobit a rozšiřovat pomocí tzv. *toolboxů*, stejně jako je tomu pro MATLAB. Jedním z dostupných rozšíření je i toolbox pro modelování, simulaci a analýzu Petriho sítí.

Je důležité zmínit, jelikož se jedná o profesionální program, že je nutné pro jeho provozování vlastnit placenou licenci. Zároveň kvůli své komplexnosti klade Simulink na uživatele potřebu investovat čas do seznámení se s nástrojem a prozkoumání uživatelského grafického rozhraní i pro realizaci jednoduchých úloh. Zmíněný problém do určité míry řeší celá řada oficiálních i neoficiálních dokumentů a video tutoriálů přítomných na internetu. Tyto materiály mají za cíl uživateli pomoct se zorientovat v dostupných nástrojích, včetně jejich používání a usnadnit tak uživateli práci s prostředím nebo ho uvést do dané problematiky.

Výhodou je i možnost spustit program plně ve webovém prohlížeči a v důsledku tak odpadá nutnost stažení a instalace produktu. Naopak nevýhodou použití online varianty je dopad na výkon, což je ovšem dáno vlastnostmi webového prohlížeče. Při používání online varianty jde znát citelná odezva grafického uživatelského rozhraní, což může být pro uživatele mírně frustrující. Stejně tak doba provedení úkonů jako je simulace nebo vykreslení výsledků do grafu, které jsou výpočetně náročné trvají déle, než je tomu v desktopové variantě.



Obrázek 2.4: Grafické uživatelské rozhraní online varianty programu Simulink.

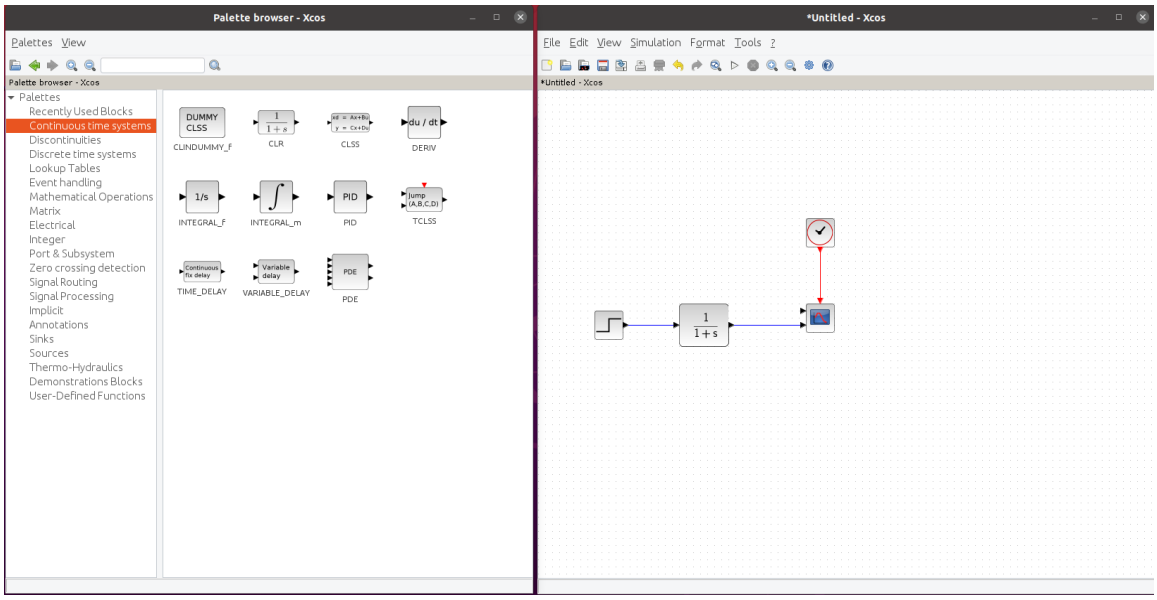
Xcos

Volně dostupnou a neplacenou alternativou je program Xcos[26] , jenž je dostupný pod licencí GPL. Stejně jako v případě předchozího produktu se jedná o nadstavbu již existujícího prostředí pro numerické výpočty Scilab. Oba produkty jsou vyvíjeny a spravovány společností *Scilab Enterprises*. Xcos je stejně jako Simulink grafický editor pro tvorbu spojitých i diskrétních modelů a sdílí spolu celou řadu vlastností jako je např. systém toolboxů pro rozšiřování poskytované funkcionality.

Xcos neposkytuje v základu tak obsáhlou knihovnu blokových schémat a nejsou zde dostupné určité pokročilé funkce jako v konkurenčním produktu Simulink, zmíněného výše 2.5. Tento fakt ale může být pro studenta nebo začátečnického uživatele softwaru tohoto typu, spíše výhodou, protože ho na první pohled neodstraší svou komplexností. Pokud nepotře-

buje konkrétní specializovanou knihovnu, která není v programu Xcos dostupná, tak může být volba tohoto produktu lepší, než volba jiných řešení. Zároveň, protože jedná o do jisté míry méně komplexní nástroj, tak do jisté míry odpadá nedostatek týkající se uživatelského rozhraní, který byl zmíněn na konci předchozího odstavce u produktu Simulink.

Scilab Enterprises se vydal odlišnou cestou a na místo poskytnutí nástroje v online verzi vytvořili cloudovou službu, jenž poskytuje rozhraní pro využívání algoritmů a prostředků platformy Scilab. [3]



Obrázek 2.5: Uživatelské rozhraní programu Xcos.

Modelica a OpenModelica

Modelica [9] je volně dostupný, objektově-orientovaný jazyk pro modelování komplexních systému ze širokého spektra různorodých oborů jako je modelování v mechatronice, automobilovém, leteckém nebo energetickém průmyslu. Jazyk byl vyvinut neziskovou organizací Modelica Association v roce 1997. Model je zapsán pomocí diferenciálních, algebraických nebo diskrétních rovnic. Jazyk Modelica slouží pouze k vytvoření modelu, nikoliv k jeho simulaci. O samotnou simulaci by se měl postarat až simulační engine, jenž na vstupu přijímá model zapsaný v tomto jazyce.

Příklad jednoduchého systému prvního řádu definovaného rovnicí $x' = (1 - x)$ v jazyce Modelica pak vypadá následovně:

```
model FirstOrder
  Real x;
equation
  der(x) = 1-x;
end FirstOrder;
```

OpenModelica [17] je open-source modelovací a simulační software založený na jazyku Modelica zmíněném výše 2.5. Produkt aktivně vyvíjí nezisková společnost *Open Source*

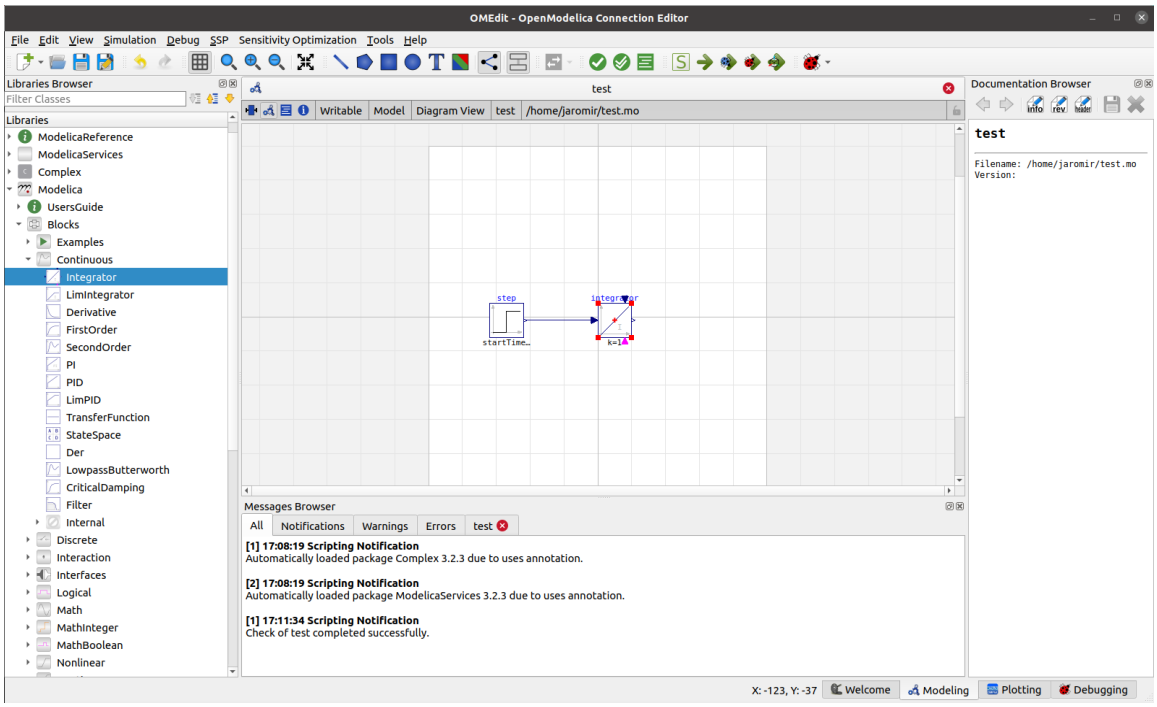
[3] https://www.scilab.org/cloud/scilab-cloud-apaas

Modelica Consortium, přičemž na vývoji se podílejí studenti, výzkumníci a vývojáři z celého světa. Díky svým vlastnostem je produkt hojně využívaný v akademickém i průmyslovém prostředí.

OpenModelica se skládá z několika nástrojů a programů, ale hlavním z nich je kompilátor OMC, jenž se stará o generování kódu z modelů zapsaných v jazyce Modelica a jejich překlad do cílového jazyka C. Tento přeložený kód pak slouží pro samotnou simulaci. O grafické uživatelské rozhraní pro tvorbu modelu pomocí bloků nebo textového zápisu, nastavování parametrů simulace a zobrazování výsledků je zodpovědný editor OMEdit. Editor nabízí mimo jiné i možnost simulaci krokovat a ladit nebo spravovat uložené modely.

Pro výukové účely je součástí sady i interaktivní rozhraní OMNotebook, které účelem připomíná pracovní sešit ze školy. OMNotebook obsahuje sadu edukační materiálů, včetně teoretických i praktických úkolů. V těchto úkolech je uživatel vyzván aby realizoval úlohu jako například vytvořit model podle jednoduchého zadání do textového pole nebo odpověděl na otázku týkající se zvoleného tématu. Odpověď uživatele je následně vyhodnocena, jestli je dobře nebo špatně. Uživatel se tak může díky těmto příkladům postupně seznámit s jednotlivými nástroji nebo se procvičit v činnosti, kterou potřebuje.

Na internetu je možné dohledat opravdu vysoké množství zdrojů jako jsou fóra, návody, manuály, video tutoriály nebo dokonce odborné přednášky pro seznámení se s prací v prostředí OpenModelica v různých doménách. To je dáno otevřeností projektu a širokou základnou uživatelů.



Obrázek 2.6: Uživatelské rozhraní editoru OMEdit.

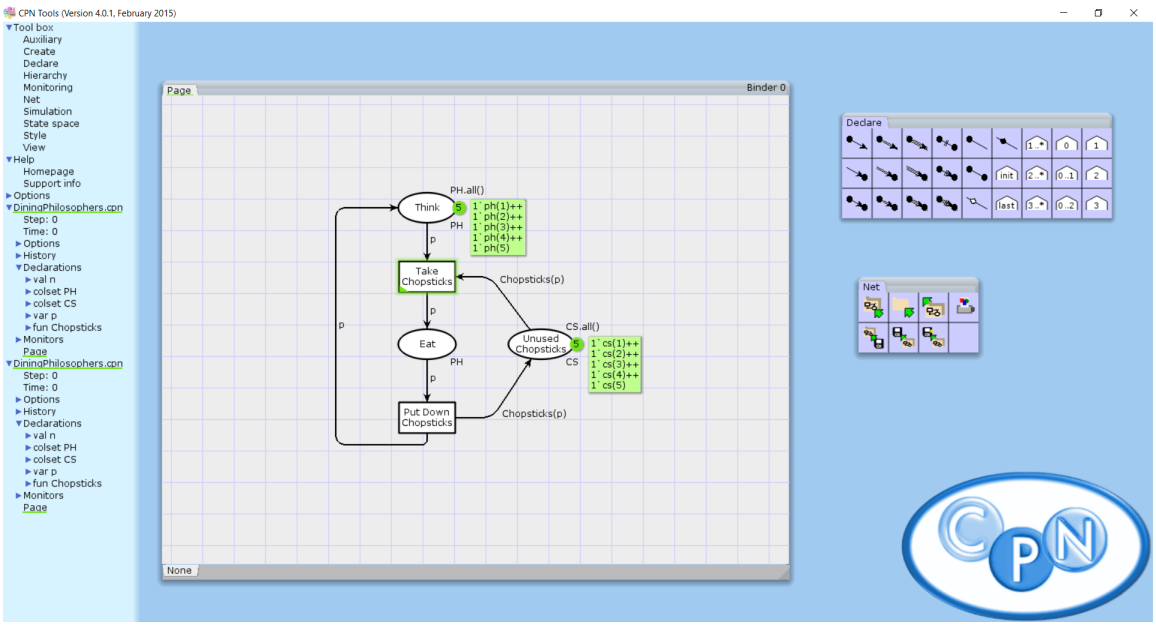
CPN Tools

Plným názvem *Colored Petri Nets Tools* [6] je volně dostupný program který poskytuje uživateli nástroje pro editování, simulaci a analýzu Petriho sítí. Důležitou vlastností, kterou software nabízí, je podpora pro časované Petriho sítě a barvené Petriho sítě. Vývoj projektu

CPNTools probíhal v letech 2000 až 2010 na dánské univerzitě ve městě Aarhus. Po roce 2010 převzala projekt skupina AIS sídlící na Unversity of Technology situované ve městě Eindhoven v Nizozemí. Program se skládá z grafického prostředí editoru a přidruženého simulátoru.

Grafické uživatelské rozhraní je řešeno poměrně atypicky. Veškeré ovládání aplikace je nutné realizovat pomocí myši. Při kliknutí pravým tlačítkem dojde k rozbalení menu kolem kurzoru, z něhož jsou pak vybírány potřebné akce. Při prvním použití aplikace bez alespoň částečného prostudování manuálu je pro uživatele téměř nemožné se v rozhraní zorientovat.

Poslední verze CPN Tools je vyvíjena pouze pro operační systém Windows. Na operačních systémech Mac a Linux je možné nástroj používat pouze při využití virtuálního prostředí. Další nevýhodou poslední verze je i nutnost mít nainstalované prostředí Java.



Obrázek 2.7: Uživatelské rozhraní CPN Tools

2.6 Použité technologie

C++

Volba jazyka C++[14] pro implementaci simulátoru je dána zadáním. Jazyk C++ je kompilovaný, silně typovaný, multiparadigmatický programovací jazyk, který vznikl rozšířením jazyku C. Jedním z podporovaných paradigmat je i objektově-orientované, jenž je použito pro návrh a implementaci simulátoru. Dalšími vlastnostmi, které jazyk nabízí jsou podpora statické i dynamické typové kontroly, typovou inferenci a širokou paletu dostupných knihoven. V praxi je jazyk C++ využíván pro tvorbu vysoce výpočetně náročného softwaru jako jsou např. operační systémy, herní enginy, webové prohlížeče, bankovní aplikace, jiné kompilátory nebo nástroje pro vědecké výpočty.

Díky tomu, že se jedná o jazyk, který je přímo překládán do binárního kódu stroje a pro jeho výkon není potřeba žádného virtuálního stroje jako je tomu např. u jazyku Java, je pro tento kód charakteristická vysoká rychlost provedení. Pro tento jazyk je dostupná podpora překlad do WebAssembly v podobě projektu Emscripten, zmíněného zde 2.2.

CMake

Pro kontrolu nad kompilací zdrojových kódů simulátoru byl použit nástroj *CMake* [1]. Nástroj využívá konfigurační soubory s názvem *CMakeLists.txt*. Tyto soubory jsou umístěny v každé složce projektu a vývojář pomocí konstrukcí jazyka dodávaného společně s nástrojem deklarativně popíše, jaké soubory jsou ve složce obsaženy a co má být výstupem jejich kompilace. *CMake* je pak schopen z těchto konfiguračních souborů vygenerovat soubory sestavení pro daný systém.

Typescript

Typescript[2] je open source programovací jazyk vyvinutý společností Microsoft v roce 2012. Jedná se o rozšíření jazyka JavaScript o statické typování, třídy, rozhraní, výčtové typy a řadu dalších vlastností, kterými standardní JavaScript nedisponuje. Protože se jedná o nadstavbu, tak platí že každý JavaScriptový kód je automaticky validním TypeScript kódem.

Kód zapsaný pomocí TypeScriptu je pomocí překladače transpilován do JavaScript kódu, který je pak interpretován prohlížečem. Statická kontrola při překladu zajistí upozornění na případné chyby jako mohou být neodpovídající typy proměnných. Díky striktní typové kontrole je docíleno lepší prevence před neočekávaným chováním.

V neposlední řadě díky přítomnosti typů poskytuje většina vývojových prostředí lepší InteliSense pro dokončování zdrojového kódu. Díky tomu je pro vývojáře programování příjemnější a rychlejší.

Pro výhody výše popsané byl tento jazyk zvolen pro vývoj frontend části projektu namísto JavaScriptu, jenž lze považovat za tradiční jazyk pro vývoj webových aplikací.

React

React [3] je deklarativní knihovna pro tvorbu uživatelského rozhraní dostupná pro jazyk Javascript i TypeScript. Za vývojem knihovny stojí společnost Facebook, která ji postupně v letech 2011 a 2012 nasadila ve svých aplikacích Facebook a Instagram. Z architektury MVC se React zaměřuje pouze na část View.

Grafické uživatelské rozhraní vytvořené pomocí Reactu se skládá z komponent. Koncepčně jsou komponenty funkce, které přijímají parametry zvané *props* a vrací HTML element, který se má zobrazit. Skládáním a kombinováním těchto komponent pak vznikne výsledné grafické uživatelské rozhraní. Komponenty mohou mít v sobě uchovanou informaci o jejich stavu. Na změnu stavu pak komponenty reagují svým překreslením.

Aby bylo vytvoření komponenty pro programátora příjemnější, poskytuje knihovna vlastní jazykové rozšíření s názvem JS, které je syntaxí podobné HTML a slouží pro definování struktury vykreslení komponenty. Jednoduchá komponenta vytvořená pomocí funkce pak může vypadat třeba následovně:

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

Recharts

Protože jedna z funkčností, které má výsledná aplikace poskytovat je i vhodná prezentace statistik simulace uživateli pomocí grafů, byla pro tento účel zvolena knihovna *Recharts*[4]. Jedná se o jednoduchou knihovnu obsahující React komponenty, pomocí nichž lze výsledný graf sestavit a definovat další vlastnosti jeho zobrazení.

Redux

Redux[8] je knihovna a stejnojmenný architektonický vzor pro správu stavu aplikace. Knihovna staví koncepčně na již existujícím architektonickém vzoru s názvem Flux.

Sklad

Základní myšlenkou knihovny je, že stav celé aplikace je uložen na jednom místě, kterému se říká *Store*. Sémantikou připomíná *Store* spíše *sklad*, než-li obchod, jak je tato část označována ve článcích v českém jazyce, proto bude ve zbytku práce použito označení sklad. Stav uložený ve skladu je v podstatě ekvivalent modelu pro architekturu MVC. Komponenty odebírají (*subscribe*) potřebná data pro zobrazení ze skladu. Data ve skladu jsou dostupná pouze pro čtení a nelze je měnit přímo a celý stav aplikace je uchován v jednom jediném JavaScriptovém objektu.

Akce

Pro změnu stavu uloženého ve skladu slouží *akce* (*actions*). Ty jsou vysílány (*dispatch*) komponentami na základě uživatelovi interakce s komponentou. Každá akce má název, jenž slouží pro odlišení jednotlivých typů akcí, a tzv. *payload*, tedy data informující o detailech akce. Těmito daty může být například klávesa kterou uživatel stiskl nebo místo na obrazovce, kde bylo provedeno kliknutí. Opět se jedná o JavaScriptový objekt, který v sobě obsahuje všechny potřebné informace.

Reduktor

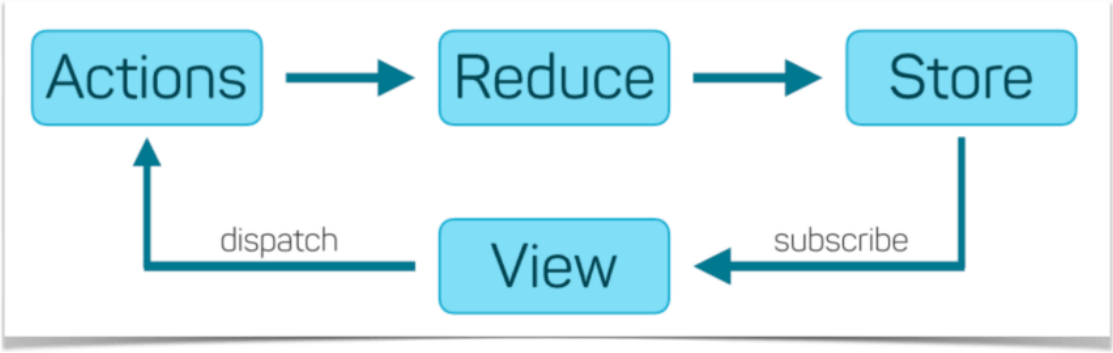
Akce pak zpracovává část zvaná *reduktor* (*reducer*). Každý typ akce zpracovává právě jeden reduktor. Reduktor není nic jiného než JavaScriptová funkce, v jejímž kódu je naprogramována logika, jak má aplikace na danou uživatelskou interakci zareagovat. Tato funkce přijímá na vstupu dva atributy. Prvním je současný stav aplikace, druhým atributem je zpracovávaná akce. Reduktor následně vypočítá nový stav aplikace a tento nový stav je použit pro aktualizaci stavu ve skladu. Na událost aktualizace stavu ve skladu pak zareagují všechny komponenty, které ze skladu odebírají data a pokud detekují, že se vybraná data změnila od předchozí aktualizace, pak vyvolají funkci pro svoje překreslení.

Proč Redux?

Volba této knihovny byla provedena až v průběhu implementace, kdy vznikla potřeba refaktorovat již vytvořený prototyp editoru, kvůli komplikovanému udržování stavu napříč komponentami. Aplikování koncepce knihovny do projektu přineslo pozitivní efekt na strukturu a přehlednost kódu, lepší oddělení logiky od komponent a s tím i větší znovupoužitelnost kódu. Funkce, které zprostředkovávaly logiku aplikace byly v prototypu rozprostřené na

[4] https://recharts.org/en-US/

vícero místech a často se nacházely přímo v kódu jednotlivých komponent. Tato logika byla následně alespoň ze značné části vyextrahována do akcí a reduktorů a umístěna do separátního souboru. Koncepce Redux architektury je znázorněno na obrázku 3.2.



Obrázek 2.8: Schéma zobrazující koncept knihovny Redux. Obrázek byl převzat z [10]
.

Sass

Pro stylování prezentační části aplikace bude použito volně dostupné rozšíření CSS zvané *Sass* [4]. Rozšíření se skládá ze stejnojmenného pre-procesoru a skriptovacího jazyku SassScript. Kód zapsaný pomocí SassScriptu je pak pomocí zmíněného pre-procesoru transpilován do CSS. Soubory používající toto rozšíření mají příponu .scss. Rozšíření vývojáři umožňuje použití vlastností, které ve standardním CSS nejsou dostupné. Těmito vlastnostmi jsou například použití proměnných, vnořená pravidla, mixiny, importy, dědičnost a mnohé další. Díky tomu je pro vývojáře správa stylů v aplikaci jednodušší.

Visual Studio Code

Pro vývoj bylo zvoleno prostředí Visual Studio Code[5]. Jedná se o integrované vývojové prostředí vyvíjené společností Microsoft, které disponuje celou řadou kladných vlastností jako je zejména dostupnost na všech platformách, jednoduché uživatelské rozhraní, možnost integrace nástrojů podporující vývoj v různých jazycích, knihovnách i technologiích. Podle průzkumu dostupného na webové stránce StackOverflow [5] se jedná o jedno z nejoblíbenějších vývojových prostředí mezi programátory za rok 2021. Zejména poslední zmíněná vlastnost byla důležitá při procesu volby prostředí, kvůli přítomnosti většího množství jazyků a knihoven ve výsledném projektu.

[5] https://code.visualstudio.com/

Kapitola 3

Specifikace a návrh řešení

V této kapitole je nejprve detailně rozebrána specifikace funkčnosti a vlastností, které jsou od aplikace, jenž má být výstupem této práce, očekávány. V další sekci jsou rozebrány důvody, jenž vedli k volbě technologií a architektonickému návrhu aplikace. Další částí jsou pak sekce obsahující návrh realizace pro jednotlivé části řešení a jejich rozbor.

3.1 Specifikace výsledné aplikace

Výsledná aplikace se bude sestávat ze dvou částí, a to editoru a simulátoru, jenž budou poskytovat funkčnost, kterou lze shrnout do následujících tří bodů:

- Tvorba modelu pomocí časované stochastické Petriho sítě nebo spojitého blokového schéma
- Simulace vytvořeného modelu
- Prezentace statistik získaných v průběhu simulace

Editor bude tvořit frontend vrstvu aplikace a měl by poskytovat uživateli prostředky pro tvorbu modelu pomocí časované stochastické Petriho sítě nebo spojitého blokového schéma. V podstatě by editor měl být jednodušší variantou editorů, které jsou součástí analyzovaných řešení v první kapitole 2.5. Důraz je kladen zejména na srozumitelné uživatelské rozhraní a jednoduchost, proto není uvažována podpora pro hierarchické strukturování modelu. Uživatel by měl na první pohled poznat, k čemu jsou jednotlivé části uživatelského rozhraní určeny a jak aplikaci používat.

Přes interakci s rozhraním editoru bude uživatel moci nad vytvořeným modelem provést simulační experiment. Za provedení simulace a sběr statistik je pak zodpovědný modul simulátoru, jemuž je předán model vytvořený v editoru. Simulátor pak předá získané statistiky zpět editoru, který je vhodnou formou zobrazí uživateli. Příklady budou dostupné pod tlačítkem v horní liště a po jeho kliknutí dojde k zobrazení sady příkladů v modálním okně. Po zvolení příkladu dojde k vytvoření modelu z příkladu v hlavní ploše, nad kterým bude moci uživatel dále pracovat.

Protože cílem práce je i demonstrovat použití technologie WebAssembly, bylo rozhodnuto, že výsledná aplikace bude řešena jako *single-page* webová aplikace. Díky tomu nebude nutné řešit podporu pro různé operační systémy, ale pouze pro webové prohlížeče. V důsledku by pak aplikace měla být dostupná na všech zařízeních, kde je možné provozovat webový prohlížeč, tedy včetně mobilních platforem, avšak hlavním cílem současného vývoje

je soustředit se primárně na použití na zařízeních jako jsou notebooky a stolní počítače. Další výhodou zvoleného přístupu je absence nutnosti instalace aplikace pro její použití. Jako hlavní prohlížeč pro vývoj byl zvolen *Google Chrome*.

3.2 Volba způsobu realizace

Výsledná aplikace by měla být složena ze dvou částí, a to editoru a simulátoru. Zadání specifikuje použití jazyku C++ pro část simulátoru, avšak volba technologie pro vytvoření editoru, jenž bude tvořit prezentační vrstvu aplikace, zůstala otevřená.

První zvažovanou variantou bylo editor vytvořit pomocí některého z C++ framweworků pro tvorbu grafického uživatelského rozhraní, které podporují přenesení do webového prostředí prostřednictvím WebAssembly. Díky tomu by byl v celém projektu využit pouze jeden programovací jazyk.

Zde se nabízelo použití framweroku Qt, jenž obsahuje podporu pro překlad do WebAssembly pomocí sady Emscripten. Na stránkách Qt je dostupný dokument[1] obsahující detaily a limitace tohoto přístupu. Na základně analýzy zmíněné možnosti vývoje, příkladů aplikací, jenž byli pomocí této metody vytvořeny, a obavám, že by se v průběhu vývoje narazilo na překážky spojené s tímto přístupem, nebyla tato možnost nakonec zvolena.

Při průzkumu článků o využití WebAssembly pro vývoj aplikací, byl nalezen článek[12] a s ním spojená aplikace, jenž obsahuje wasm modul pro syntetické generování zvuku. V této aplikaci je využita kombinace knihovny React pro prezentační vrstvu a WebAssembly pro výpočetně náročnou část. Tento případ užití je ostatně zmíněn i v sekci věnované WebAssembly v první kapitole 2.1. Zmíněný článek a aplikace sloužily jako inspirace pro použité technologie a způsob řešení.

Následně tedy bylo rozhodnuto editor a zbytek grafického uživatelského rozhraní aplikace řešit pomocí standardních technologií pro vývoj webových aplikací jako jsou HTML, CSS a JavaScript. Simulátor pak bude do aplikace integrován pomocí adaptéru jako WebAssembly modul. Při tomto přístupu bude editor a simulátor od sebe oddělen na nezávislé části.

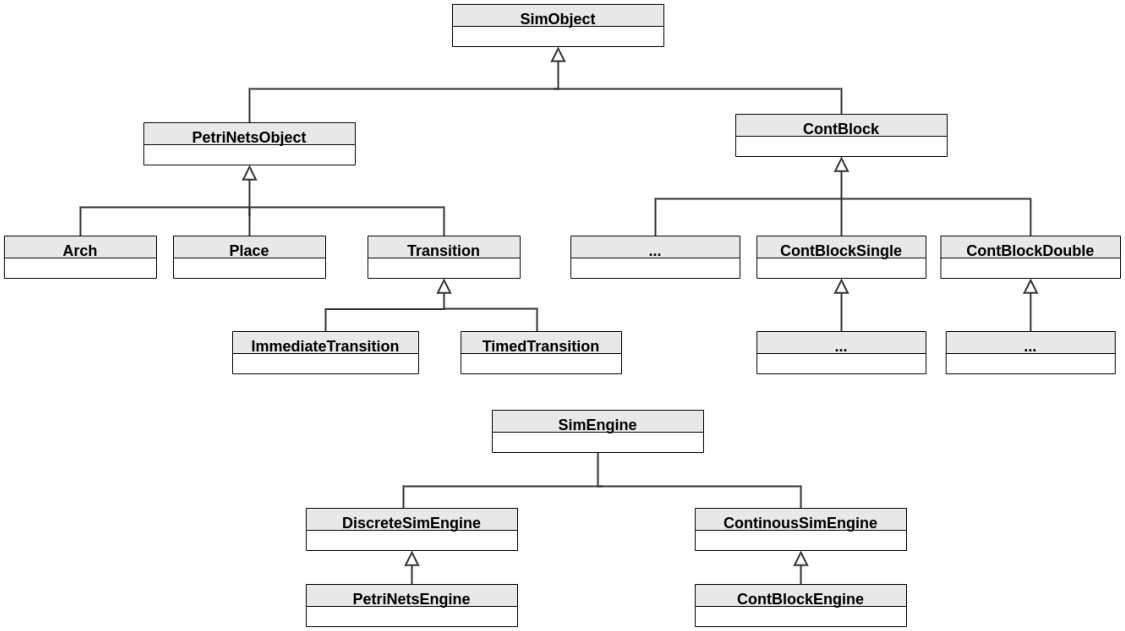
3.3 Návrh simulátoru

Modul simulátoru bude řešen jako samostatná knihovna, obsahující potřebné datové struktury a algoritmy pro vytvoření modelu v požadované oblasti a jeho následné simulace. Pro vytvoření modelu budou sloužit třídy odpovídající modelovaným entitám daného oboru problematiky. Pro Petriho sítě to znamená vytvoření tříd pro reprezentování míst, hran, okamžitých přechodů a časovaných přechodů. Pro simulaci blokových schémat budou v knihovně přítomny třídy pro bloky reprezentující základní matematické operace jako jsou sčítání, odčítání, násobení, dělení a tak podobně. Dále zde bude třída pro spojitý blok integrátoru.

O průběh simulace se budou starat třídy s příponou `engine`. Třídy této kategorie budou sloužit jako kontejner sdružující všechny objekty, které jsou součástí modelu a budou obsahovat metodu *simulate*, která pak bude sloužit k řízení celé simulace. Pro řízení diskrétní simulace bude sloužit třída *DiscreteEngine*, jenž bude pro řízení simulace implementovat *next-event* algoritmus. Z této třídy pak děděním vznikne třída *PetriNetsEngine* pro simulaci Petriho sítí. Identický postup bude aplikován i pro simulaci blokových schémat. Řízení spojité simulace bude obsluhovat třída *ContinousSimEngine*, ze které děděním vznikne třída

[1] https://doc.qt.io/qt-5/wasm.html

ContBlockEngine pro simulaci blokových schémat. Další zodpovědností tříd bude sběr statistik. V případě Petriho sítí budou pro místa sbírány informace o počtu tokenů v čase a pro přechody pak počet provedení přechodu. Pro spojité blokové schéma pak bude engine zaznamenávat hodnoty stavové proměnné integračních bloků.



Obrázek 3.1: Diagram hierarchie tříd simulátoru

Sestavení do WebAssembly

Pro kontrolu nad sestavením knihovny bude použit nástroj CMake. Díky tomu bude možné využít pro přenesení celého projektu s knihovnou do WebAssembly program emcmake zmíněný v sekci věnované Emscripten. Obdobně bude použit nástroj Embind pro vytvoření vazeb potřebných tříd a struktur tak, aby byla knihovna použitelné jako WebAssembly modul ve výsledné webové aplikaci. Součástí projektu budou skripty s příkazy pro překlad a práci s projektem.

Další vlastnosti simulátoru

Při řešení bude kladen důraz na strukturování kódu tak, aby byl dostatečně dobře rozšiřitelný, přehledný a udržitelný. V konkrétním důsledku by pak část zodpovědná za simulaci blokových schémat měla být dostatečně jednoduše rozšiřitelná o podporu pro další typy bloků. Zároveň by výsledná knihovna měla být realizována tak, aby ji bylo možné použít i mimo tento projekt. Využití této knihovny by mohlo být např. ve studentských projektech.

3.4 Návrh editoru

Návrh editoru byl inspirován editory analyzovaných řešení zmíněných v první kapitole 2.5 a webovou aplikací pro tvorbu diagramů draw.io[2]. Grafické rozhraní editoru se bude skládat ze tří hlavních komponent.

Prvním z těchto komponent bude plocha, ve které se budou nacházet diagramy objektů modelu a ve které s nimi bude moci uživatel pracovat. Tato plocha bude řešena pomocí SVG HTML elementu. Pomocí myši bude možné diagramy označovat a pohybovat nimi po ploše. Po označení diagramu dojde ke zvýraznění tzv. *endpointů*, což budou body na okrajích diagramu. Tyto body budou sloužit jako koncové body spojovacích hran. U bodů, z nichž bude moc být spojení vedeno, bude přítomno tlačítko, po jehož stisknutí dojde k vytvoření spojovací hrany z daného bodu zakončené šipkou. Tato hrana bude sloužit ke spojování mezi *endpointy* jednotlivých diagramů.

Další částí bude menu, které bude sloužit pro přidávání objektů do modelu. Při přidání objektu do modelu je přidán i diagram odpovídající danému typu objektu do hlavní plochy. V menu budou vždy přítomny tlačítka pro přidání objektů pouze z vybrané oblasti. Tuto oblast určí uživatel v hlavním menu aplikace. Díky tomu bude menu přehlednější, nebude zbytečně přeplněné a matoucí pro uživatele.

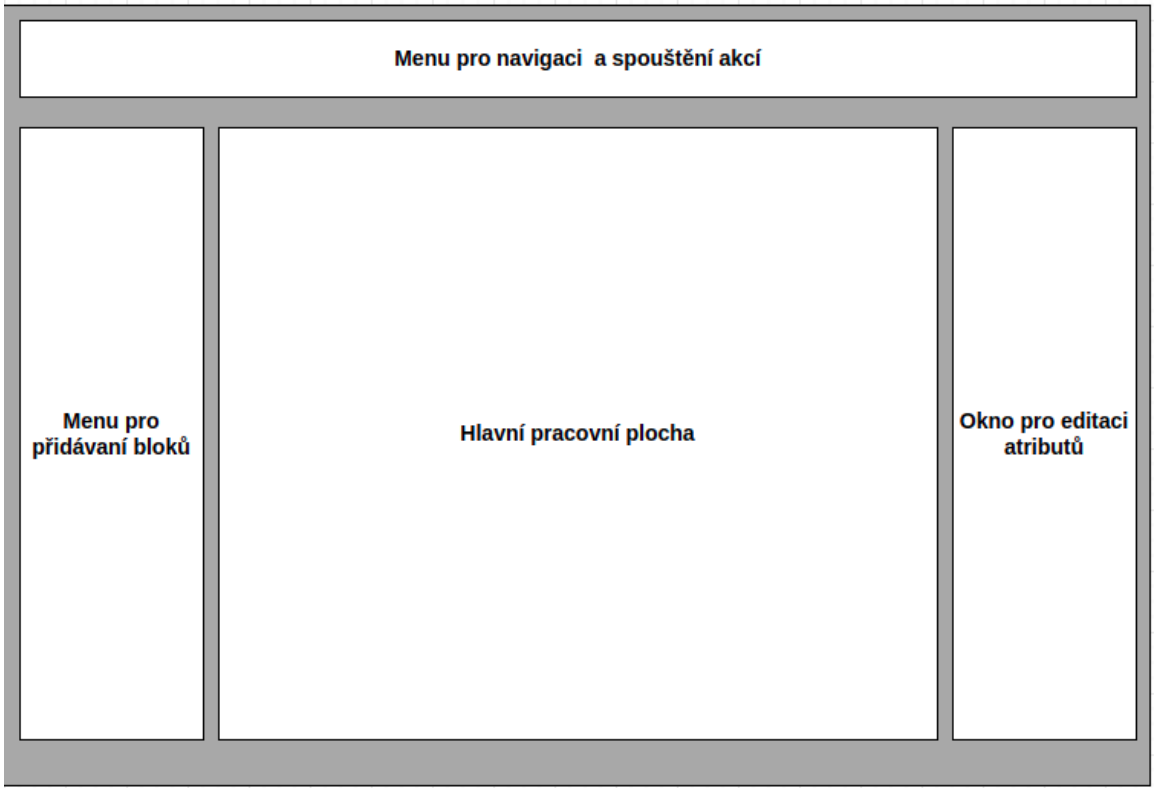
Poslední částí editoru bude okno pro editaci atributů objektů. Na událost označení některého z digramů toto okno zareaguje zobrazením editačního okna pro daný objekt. Zobrazené okno se bude lišit podle označeného typu objektu. Pokud není označen žádný objekt, potom je zobrazeno okno pro editaci parametrů simulace.

V horní části grafického rozhraní se bude nacházet menu pro navigování v aplikaci a provádění požadovaných operací jako je spuštění simulace nebo zobrazení panelu s příklady.

Vizualizace výsledků simulace

Po provedení simulace přejde editor do módu pro zobrazení statistik. Při přechodu změní rozvržení hlavního okna tak, že skryje boční menu pro přidávání bloků, hlavních plocha se přesune mírně doleva na nově vzniklé místo a do pravé části obrazovky se umístí okno pro zobrazení statistik. V tomto okně jsou pak zobrazeny statistiky pro označený element v hlavní ploše.

[2] https://app.diagrams.net/



Obrázek 3.2: Návrh grafického rozhraní editoru.

Kapitola 4

Implementace

Tato kapitola je věnována procesu implementace. Čtenář se zde dozví o důležitých částech, jenž tvoří výslednou aplikaci. Úvod je věnován založení projektu a struktuře složek. Následuje sekce s detaily důležitých částí implementace editoru jako jsou vytvořené komponenty grafického uživatelského rozhraní, správa stavu aplikace a obsluhu uživatelských interakcí. Pro část simulátoru jsou zmíněny důležité třídy a algoritmy použité pro řízení simulace.

4.1 Založení projektu

Nejdříve bylo nutné založit složku s projektem. Protože bylo rozhodnuto že frontend aplikace bude tvořen v Reactu a Typescriptu, byl projekt založen pomocí nástroje *Create React App*[1]. Nástroj poskytuje možnost vytvořit projekt rovnou s podporou pro TypeScript pomocí specifikování šablony. Nástroj byl volán následujícím příkazem: *npx create-react-app <nazev-projektu> –template typescript*.

Kromě vygenerování složky s projektem a vytvoření základní struktury složek má použití tohoto nástroje další přidanou hodnotu v podobě automatické konfigurace projektu. Další výhodou je, že součástí projektu jsou automaticky i skripty pro překlad projektu a spuštění aplikace. Pro sestavení aplikace stačí zadat do konzole příkaz *npm run build*. Při zadání příkazu *npm start* dojde ke spuštění vývojového webového serveru s aplikací. K aplikaci je pak možné přistoupit po zadání adresy *localhost:3000* do webového prohlížeče.

Struktura projektu

Vygenerovaná struktura projektu byla následně upravena do následující podoby:

```
build
docs
node_modules
pkg
public
src
    Editor
    Simulator
    App.tsx
```

[1] https://github.com/facebook/create-react-app

Soubor *App.tsx* obsahuje stejnojmennou funkci, která slouží jako hlavní grafická komponenta editoru. Přidáváním dalších komponent do této funkce je pak složeno výsledné uživatelské grafické rozhraní. Ve složce *Editor* se nachází veškeré zdroje modulu editoru jako jsou např. jednotlivé komponenty grafického rozhraní, a obdobně ve složce *Simulator* se nachází knihovna pro simulaci.

4.2 Implementace editoru

Globální stav

Jak již bylo zmíněno v sekci 2.6 v předchozí kapitole, pro uchování globálního stavu aplikace je použita knihovna Redux. Sklad se nachází v souboru Store.tsx. Díky použití Redux Toolkitu je tento stav rozdělen na tzv. řezy (*slices*). Tyto řezy se nachází ve složce Feature. Každý z řezů slouží k poskytování určité dílčí funkčnosti. Nejdůležitějším z nich je *SimObjectManagementSlice.tsx*. V tomto řezu jsou obsaženy reduktory a s nimi spojené akce pro přidávání, odebírání a modifikování objektů uložených ve skladu, jejich posun v hlavní ploše a spojování.

Model

Ve složce Model se nachází třídy odpovídající jednotlivým elementům z Petriho sítí a blokových schémat. Potom, co uživatel klikne v menu na přidání elementu daného typu, je následně vytvořena instance odpovídající třídy, která je posléze přidána do skladu. Zde bylo nutné zvolit poměrně krkolomný postup, protože limitací knihovny Redux je, že se ve skladu musí nacházet pouze serializovatelné objekty, což ale objekty vytvořené jako instance tříd nesplňují. Tento problém byl vyřešen tak, že bylo zavedeno rozhraní *ISerializable*, jehož definice obsahuje jedinou metodu *toSerializableObj*, jejíž návratovou hodnotou je serializovatelný klon dané instance. Každá třída, jejíž instance je potřeba držet ve skladu pak musí implementovat zmíněné rozhraní.

Hlavní plocha

Hlavní plochu editoru tvoří komponenta *Canvas*. Plocha je řešena, jak již bylo uvedeno v návrhu pomocí pomocí SVG HTML elementu. Do podstromu komponenty jsou pak přidávány komponenty reprezentující elementy modelu.

Mapování komponent

Mapování grafických komponent pro reprezentování jednotlivých typů objektů byl použit návrhový vzor *Factory*[18]. Pro každou oblast pak byla implementována vlastní továrna. Pro Petriho sítě je to *PetriNetsComponentFactory* a pro bloková schémata *ContBlockComponentFactory*. Funkce těchto továren přijímají na vstupu typ objektu a vrací trojici komponent. Prvním komponenta slouží pro reprezentování objektu v hlavní ploše editoru. Druhá komponenta je zobrazena v editačním okně při označení objektu a třetí pak v okně výsledků simulace. Pokud pro daný typ objektu není nutné v některém z oken nic zobrazovat, protože například nemá žádné editovatelné atributy, je jednoduše použita prázdná komponenta. To, která továrna má být použita, je dáno podle toho, která oblast je uživatelem zvolena v hlavním menu aplikace. Mapování pomocí továren probíhá pro každý objekt z modelu, který je uložen ve skladu.

Komponenty pro hlavní plochu

Název komponent, jenž mají být použity v hlavní ploše končí příponou SVG. Pro snadnější implementaci tohoto typu komponent byl vytvořen vlastní React Hook s názvem *useSVGComponentUtils*. Hook slouží k extrahování logiky z komponenty do speciální funkce a díky tomu ji lze použít i v dalších komponentách. Funkce *useSVGComponentUtils* vrací prostředky jako jsou stavové proměnné a *handlery*, díky kterým je s komponentou možné pohybovat pomocí myši v hlavní ploše, označovat ji a obstará provázání s objektem ve skladu, který daná komponenta reprezentuje.

Komponenty pro editaci atributů

I pro komponenty určené k editaci atributů objektu je zavedena konvence pro pojmenování, a to zakončení příponou *Edit*. Tento typ komponent slouží v podstatě jako formulář, který je zobrazen v pravé části grafického uživatelského rozhraní při označení objektu v hlavní ploše. Formulář se skládá z elementů pro editaci jednotlivých atributů objektu. Při změně hodnoty některého z elementů komponenta okamžitě vyvolá pomocí funkce *dispatchChangeObject* akci informující o události změny některého z atributů objektu. Následně dojde k aktualizaci příslušného objektu ve s skladu a překreslení všech komponent, kterých se změna dotkla.

Adaptéry pro modul simulátoru

Protože editor a simulátor jsou dvě oddělené části vytvořené v různých technologiích, je nutné tyto dvě části nějak propojit. Pro tento účel slouží třídy s příponou *adapter*, které jak už samotná přípona napovídá slouží jako adaptér propojující obě části. Pro Petriho sítě a blokové schéma byl implementován vlastní adaptér a použitý adaptér je zvolen obdobně jako u továren komponent s uživatelovou volbou v hlavním menu. Po kliknutí na akci spuštění simulačního experimentu předá editor potřebná data uložená ve stavu adaptéru, jako jsou objekty modelu a parametry simulace. Adaptér se pak stará o vytvoření simulačního modelu pomocí prostředků z wasm modulu simulátoru. Simulační model je posléze spuštěn. Výsledkem jsou získané statistiky experimentu, které je nutné nejdříve převést na normální JavaScript objekty, aby mohli být umístěny do skladu a následně použity pro vytvoření grafů.

4.3 Implementace simulátoru

Diskrétní simulace

Prostředky pro diskrétní simulaci jsou obsaženy ve složce *DiscreteSimulation*. O provedení simulace se stará třída *DiscreteEngine*, která pro řízení simulace využívá algoritmus Next-Event popsaný níže.

Next-event

Next-event [23] algoritmus slouží pro simulování diskrétních systémů. Algoritmus využívá struktury zvané *kalendář*[23], která slouží pro uchování záznamů o plánovaných nadcházejících událostech (*events*). Struktura reprezentující kalendář bývá nejčastěji implementována jako prioritní fronta. Záznamy v tomto kalendáři pak obsahují informaci o aktivačním čase a prioritě události. Struktura kalendáře umožňuje vkládat, odebírat a vybrat záznam s nejbližším aktivačním časem.

Pseudokód algoritmu Next-Event pak vypadá následovně:

```
Inicializace modelu, času, kalendáře...
while Kalendář není prázdný do
    Vyjmi záznam z kalendáře
    if Aktivační čas události je větší než koncový čas simulace then
        Konec simulace
    end if
    Nastav čas na aktivační čas události
    Proveď událost v záznamu
end while
```

Simulace Petriho Sítí

O simulaci Petriho sítí se pak stará třída *PetriNetsEngine*, která dědí ze třídy *DiscreteEngine* a tím přebírá algoritmus řízení simulace. Díky tomu bylo nutné pouze doimplementovat logiku průběhu simulace, o kterou se stará třída *Transition* reprezentující přechod. Třída *Transition* má přístup ke struktuře kalendáře, která je součástí *enginu*, a podle potřeby přidává a odebírá událostí svého provedení. Průběh simulace je pak možné popsat následovně:

- Při inicializaci modelu všechny proveditelné přechody naplánují událost svého provedení.
- Postupným zpracováním těchto událostí je pak prováděna samotná simulace.

Provedení přechodu

Provedení přechodu se skládá ze dvou kroků, v prvním jsou tokeny odebrány ze vstupních míst a v tom druhém přidány do výstupních míst. Po provedení jednotlivých kroků přechod informuje ostatní přechody, kterých se modifikace nějak dotkne, aby se přeplánovaly. Při přeplánování přechodu jsou oba kroky od sebe odlišeny. V průběhu prvního kroku ovlivněné přechody odebírají z kalendáře události svého provedení, a naopak v kroku druhém, kdy jsou tokeny přidávány do míst, události do kalendáře přidávají.

Okamžité přechody

Okamžité přechody přidávají do kalendáře událost svého provedení s nulovým zpožděním. Pokud je přechod proveditelný vícekrát, naplánuje událost tolikrát, kolikrát je proveditelný.

U okamžitých přechodů je možné modifikovat jejich prioritu. Tato priorita pak ovlivňuje pořadí záznamů v kalendáři, které mají stejný aktivační čas. Záznamy s vyšší prioritou jsou umístěny před záznamy s nižší prioritou.

Časované přechody

U časovaných přechodů je možné definovat dva typy zpoždění, a to buď konstantní nebo jako výsledek funkce generující exponenciální rozložení. Pro generování náhodných čísel a exponenciálního rozložení byl implementován vlastní generátor. Rozdíl oproti okamžitému přechodu je ten, že událost provedení přechodu se naplánuje do kalendáře s daným zpožděním. Po celou dobu čekání musí být přechod stále proveditelný. Pokud je přechod

proveditelný vícekrát, platí to stejné co u okamžitého přechodu. V případě že dojde k situaci, že je přechod naplánován vícekrát a dojde ke snížení kolikrát je přechod proveditelný, jsou události provedení odebrány náhodně.

Spojitá simulace

Pro spojitou simulaci slouží abstraktní třída *ContinousSimEngine*. Algoritmus řízení spojité simulace odpovídá následujícímu pseudokódu:

Inicializace modelu
while čas < koncový čas **do**
    Proveď krok simulace
end while

Simulace spojitých blokových schémat

Pro simulaci spojitých blokových schémat slouží třída *ContBlockEngine*, která dědí algrotimus řízení spojité simulace ze svého předka *ContinousSimEngine*. Následně bylo nutné pouze definovat logiku simulace do metody, která je volaná jako krok simulace. Tato metoda je v předkovi deklarována jako virtuální. Krok simulace se sestává z následujících bodů, jenž byly převzaty z kapitoly 6. studijní opory [23] :

1. Vyhodnocení vstupů všech integrátorů.
2. Provedení numerické metody.
3. Posunutí v čase o daný krok.

Numerické metody

Bloky integrátorů používají pro svoje vnitřní fungování numerickou metodu. Tato metoda je přítomna jako funkce v simulačním enginu a je možné ji předat jako parametr konstruktoru. Jako základní numerická numerická metoda byla implementována Eulerova metoda.

Překlad do WebAssembly

Knihovna je překládána do WebAssembly modulu s následujícími přepínači ovlivňující překlad:

- WASM=1 - výstupem má být wasm modul.
- MODULARIZE=1 - umožní vytvoření instance modulu až za běhu aplikace pomocí funkce.
- EXPORT_ES6=1 - umožní použití JavaScript příkazu import pro práci s modulem .
- SINGLE=FILE=1 - tento přepínač způsobí, že výstupem překladu bude pouze jeden JavaScript soubor, ve kterém je druhý .wasm soubor uložen jako base64 string. Použití tohoto přepínače se ukázalo jako kritické, kvůli problémům s načítáním samotného .wasm souboru během běhu.
- ENVIRONMENT="Web"- určuje, že modul bude použit ve webovém prostředí. Překladač díky tomu provede optimalizační kroky vedoucí k menší velikosti výsledného souboru.

4.4 Dostupnost

Jednou z důležitých vlastností výsledné aplikace je její jednoduchá dostupnost na internetu, proto bylo nutné zvolit platformu, jenž bude webovou stránku s aplikací hostovat. Protože pro správu repozitáře s projektem byla využita platforma *GitHub*[2], která zároveň poskytuje možnost pro hostování projektů na webu prostřednictvím služby *GitHub Pages*[3], byla volba směřována tímto směrem.

Samotné publikování bylo realizováno prostřednictvím balíčku *gh-pages*[4]. Tento balíček přidává do projektu skripty, pro publikaci a správu hostované stránky a je přímo určen pro práci s *GitHub Pages*. Díky tomu byl samotný proces velmi jednoduchý. Po publikaci bylo nutné provést několik málo oprav, protože skript pro publikování používá verzi sestavení projektu optimalizovanou pro produkční prostředí. Výsledkem je, že aplikace je dostupná na adrese: `https://trautenberk.github.io/edu-sim-hub/`

4.5 Testování

UnitTesty

Pro část simulátoru byla vytvořena sada jednotkových testů. Primárním účelem testů je kontrola funkčnosti stěžejních metod tříd jako například výstup funkce jednotlivých bloků. Do testů byly zahrnuty scénáře simulace jednoduchých modelů, u kterých je kontrolováno, jestli bylo dosaženo očekávaného výsledku. Spuštění testů je možné realizovat pomocí skriptu *test.sh* ve složce *Simulator*.

Testování editoru

Pro modul editoru nebyly implementovány žádné automatické testy a veškeré testování bylo provedeno manuálně. Během manuálního testování byla snaha pokrýt předpokládané scénáře, jak uživatel bude s aplikací pracovat. Chyby nalezené během testů byly opraveny, avšak to neznamená, že ve výsledné aplikaci nejsou další žádné chyby přítomny. Výsledky příkladů slouží také jako kontrola výsledné aplikace, neboť očekávané výsledky jsou předem známé.

[2] `https://github.com/`
[3] `https://pages.github.com/`
[4] `https://github.com/tschaub/gh-pages`

Kapitola 5

Příklady modelů

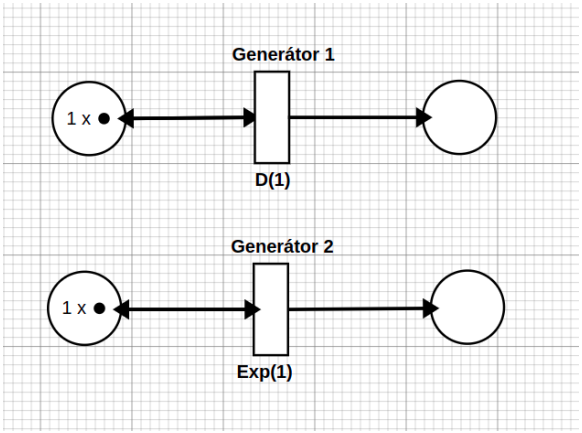
Následující kapitola se věnuje sadě příkladů, jenž je součástí aplikace. Zadání bakalářské práce specifikovalo vytvoření sady o 10 příkladech, jejichž účelem je jednak ukázat elementární vlastnosti vybraných oborů, ale slouží také jako demonstrace použití aplikace.

5.1 Příklady pro Petriho Sítě

V případě příkladů Petriho sítě je vhodnější, když si je sám uživatel vyzkouší v aplikaci, proto je vždy přítomný pouze obrázek se schématem modelu a nejsou přiloženy žádné konkrétní výsledky simulačních experimentů.

Ukázka generátorů událostí

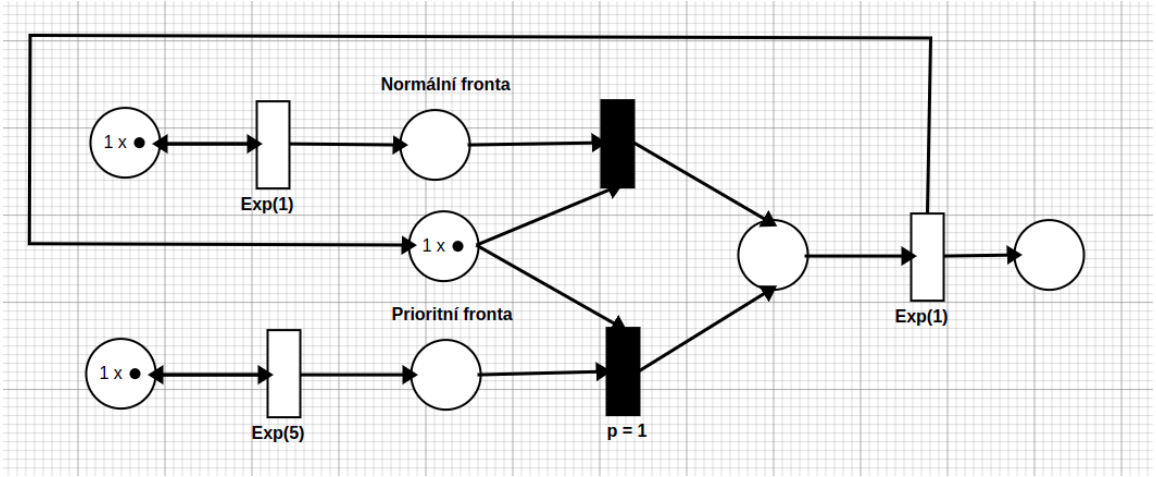
Cílem prvního z příkladů je ukázat, jak lze modelovat pomocí prvků Petriho sítě výskyt periodických a náhodných událostí v systému pomocí tzv. generátoru. Konstrukci generátoru lze využít např. pro modelování událostí příchodů zákazníků nebo výskytu chyby v systému hromadné obsluhy. Schéma příkladu je možné vidět na obrázku 5.1. Po provedení simulace příkladu je možné v získaných statistikách vidět, jak každý z generátorů v daném intervalu přidává tokeny na svoje výstupní místo, kde počet tokenů postupně narůstá, ale také že vrací spotřebovaný token na své vstupní místo, díky čemuž pak dochází k jeho opětovnému povolení.



Obrázek 5.1: Příklad pro ukázku generátorů.

Prioritní fronta

Ve druhém příkladu je ukázáno, jak může vypadat model systému, ve kterém je obsažena fronta s prioritní obsluhou. Z výsledků získaných simulačním experimentem s různými parametry je možné vidět, jak se vyvíjí počet tokenů v místech reprezentující jednotlivé fronty. Jak vypadá model příkladu v aplikaci lze vidět na obrázku 5.2.



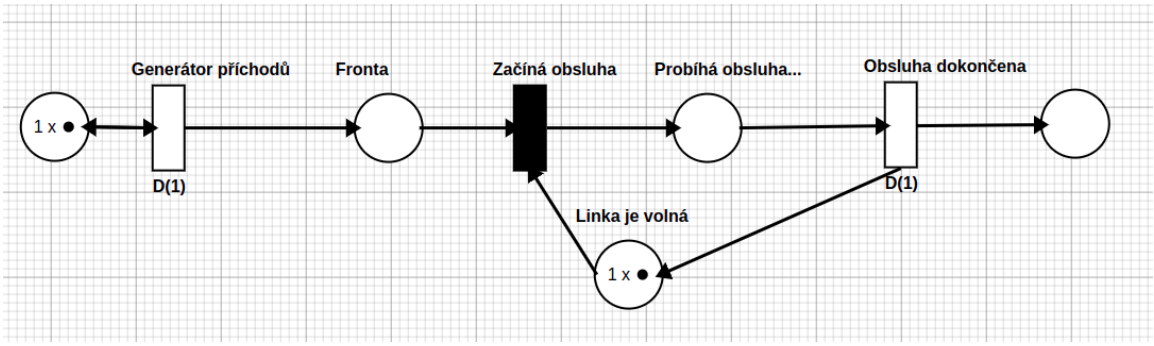
Obrázek 5.2: Příklad s prioritní frontou v editoru.

Systémy hromadné obsluhy D/D/1 a M/M/5

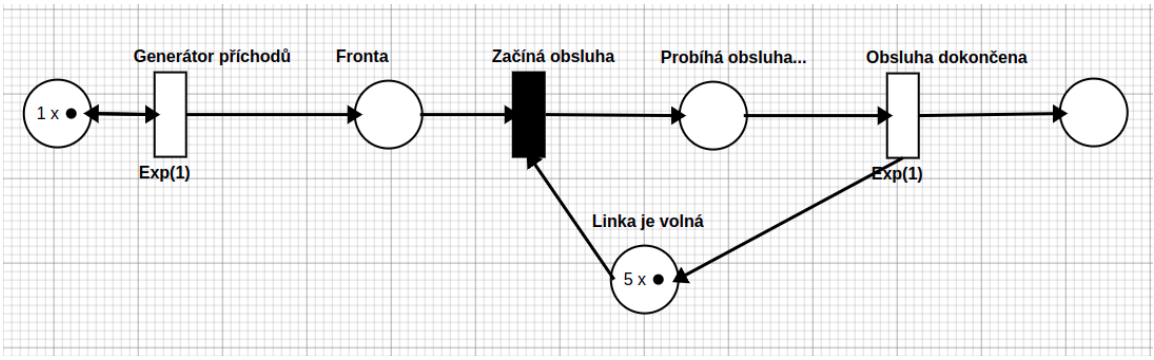
Třetí a čtvrtý příklad slouží jako ukázka modelů systému hromadné obsluhy podle odpovídající Kendalovy klasifikace. Oba příklady obsahují generátor pro modelování příchodu zákazníků a obslužné linky. Rozložení použité pro generátor a pro dobu obsluhy pak odpovídá klasifikaci systému, tedy pro model systému D/D/1 je použito konstantní a v druhém modelu je použito exponenciální rozložení. Experimentováním s modelem pak uživatel může zjistit, jaký vliv na systém má dopad změny jednotlivých parametrů. Pozorovanými aspekty zde mohou být délka fronty nebo celková vytíženost linky. Schéma obou příkladů je možné vidět na obrázcích 5.3 a 5.4.

Systém M/D/1 s výskytem poruchy

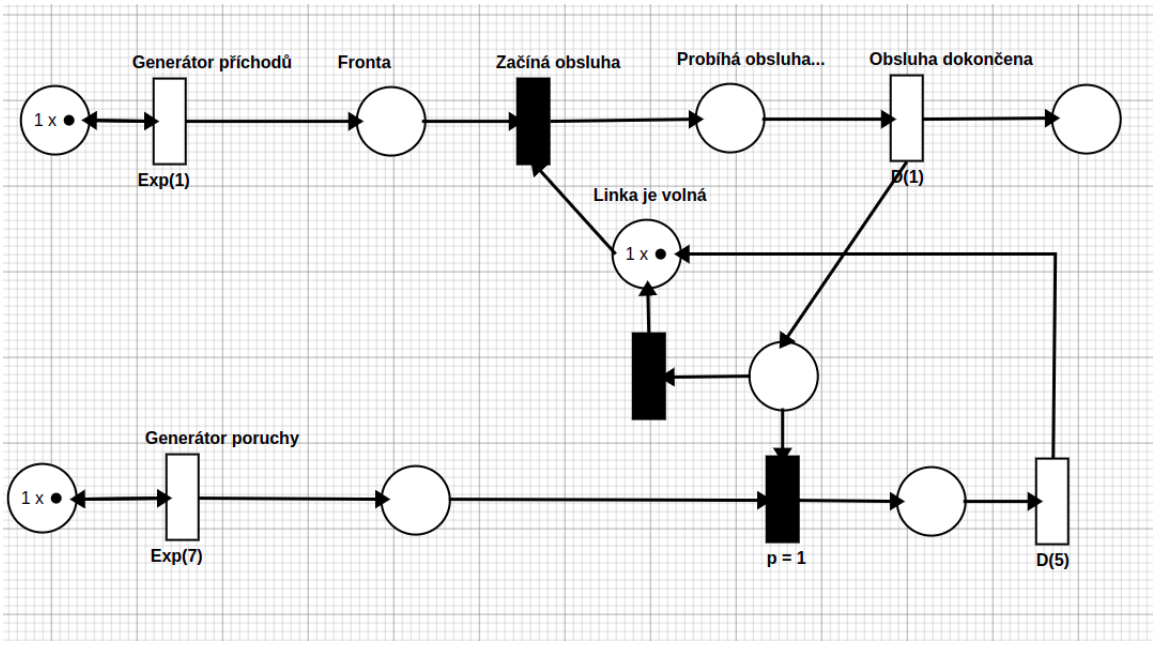
V pořadí čtvrtý příklad pro Petriho sítě má za cíl ukázat model systému, ve kterém se v určitém časovém vyskytuje událost reprezentující poruchu. Potom co se v modelu vyskytne porucha, nepřeruší se probíhající obsluha, ale po jejím dokončení je linka odstavena na konstantní dobu, než může začít další obsluha. Při změně parametrů je možné ve výsledcích experimentů pozorovat dopad poruchy na modelovaný systém. Schéma příkladu je dostupné na obrázku 5.5.



Obrázek 5.3: Příklad s modelem systému D/D/1.



Obrázek 5.4: Příklad s modelem systému M/M/5.



Obrázek 5.5: Příklad s modelem systému s výskytem poruchy.

5.2 Příklady pro spojitá bloková schéma

Úvodní příklady

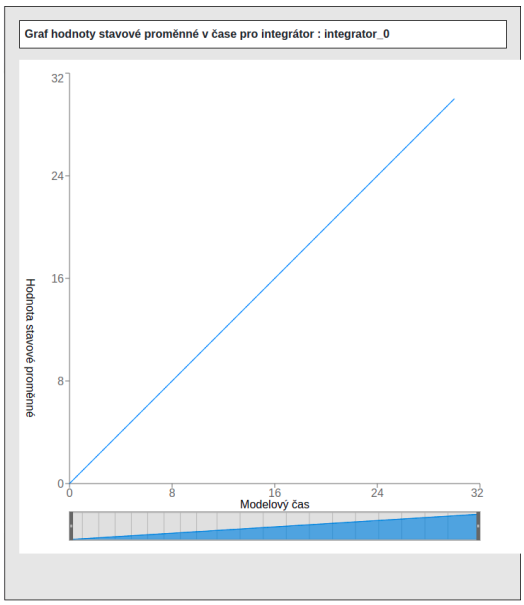
Úvodní dva příklady demonstrují řešení elementárních rovnic a neobsahují žádnou komplexní záležitost. Slouží spíše jako důkaz základní funkčnosti aplikace. Řešeny jsou následující dvě rovnice

$$f(x) = x \qquad f(x) = x^2 \tag{5.1}$$

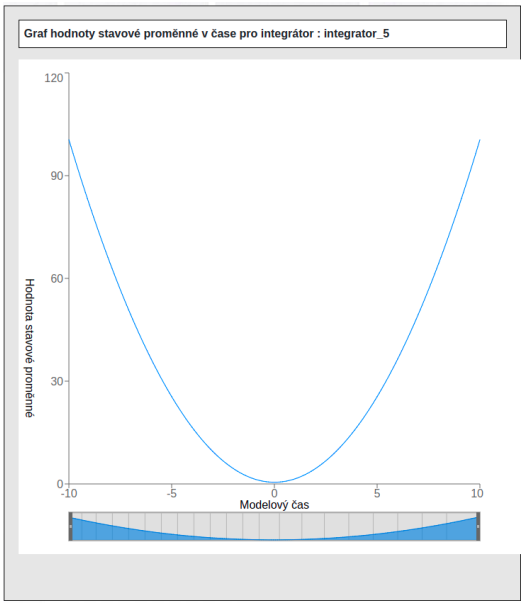
Rovnice byly upraveny do následujících tvarů, pro které byly vytvořeny odpovídající modely:

$$\begin{aligned} f(x) &= \int (x)' & f(x) &= \int (x^2) \\ f(x) &= \int 1 & f(x) &= \int 2x \end{aligned} \tag{5.2}$$

Křivky získané simulací modelů lze vidět na obrázcích 5.6 a 5.7.



Obrázek 5.6: Křivka získaná simulací prvního příkladu.



Obrázek 5.7: Křivka získaná simulací druhého příkladu.

Kruhový test

Kruhový test popisuje následující jednoduchá diferenciální rovnice:

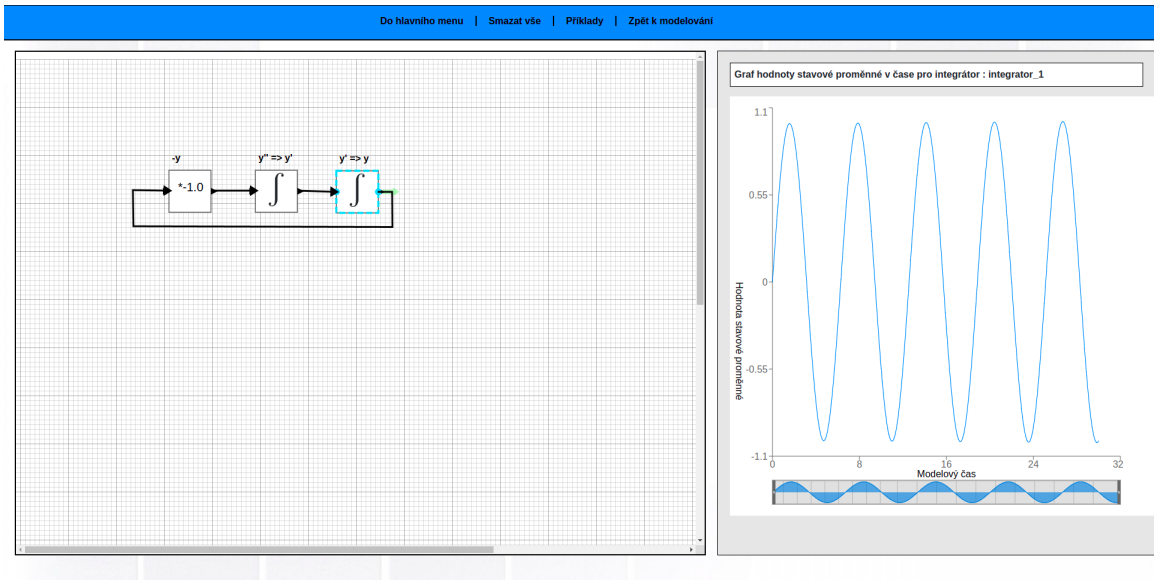
$$y''(t) + ky(t) = 0 \qquad y(0) = 0, y'(0) = |k| \tag{5.3}$$

Při dosazení $k = 1$ pak dostáváme analytické řešení: $sin(t)$. Pomocí metody snižování řádu derivace lze rovnice přepsat do následující soustavy rovnic, kterou popisuje blokové schéma

v příkladu:

$$y^{''} = -ky$$
$$y^{'} = \int y^{''} dt$$
$$y = \int y^{'} dt$$

Model pro $k = 1$ i výsledky simulace příkladu je možné vidět na obrázku 5.8



Obrázek 5.8: Schéma a výsledky modelu kruhového testu.

Van der Polův Oscilátor

V příkladu číslo čtyři je zpracován model nelineárního systému Van Der Polova oscilátoru. Příklad byl převzat z [19]. Rovnice popisující systém je následující:

$$x^{''} + x = a(1 - x^2)x^{'} \tag{5.4}$$

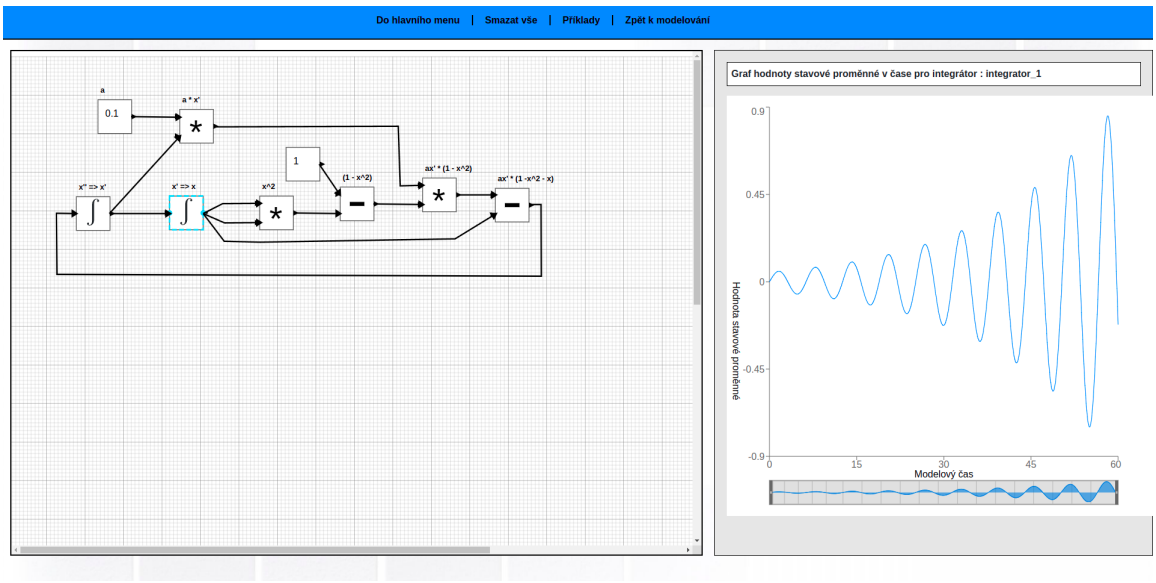
kde a je parametr. Model a výsledky simulace pro hodnoty: $a = 0.1, x^{'}(0) = 0.05, x = 0$ je pak možné vidět na obrázku 5.9.

Závaží na pružině

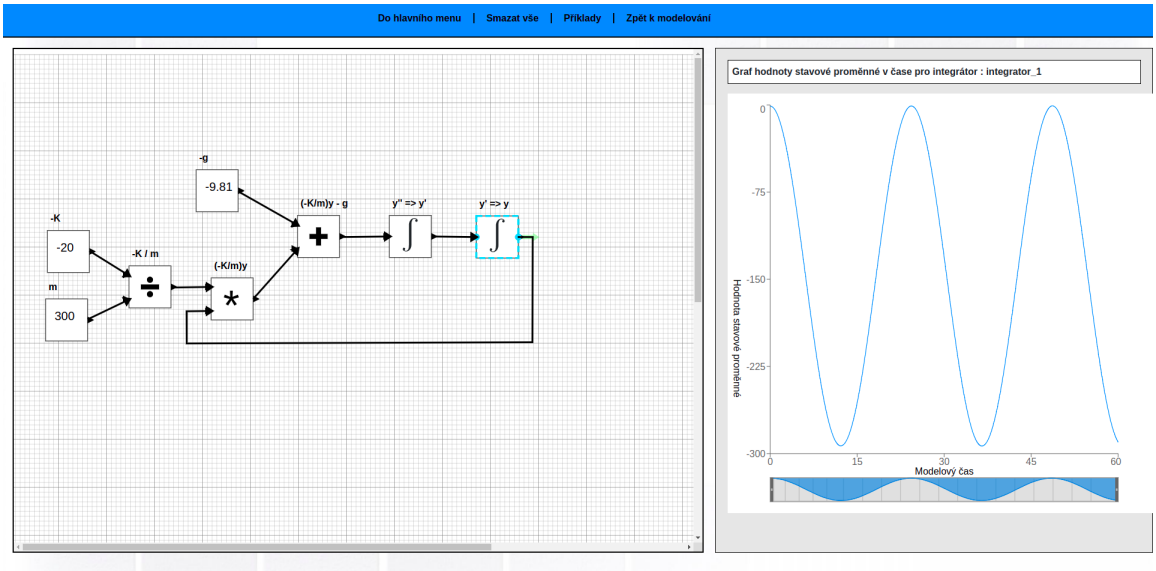
V posledním příkladu byl zpracován model systému závaží na pružině. Netlumené kmity závaží na pružině popisuje následující diferenciální rovnice:

$$y'' + \frac{k}{m}y = 0 \tag{5.5}$$

V této rovnici označuje y polohu v čase, m je hmotnost závaží, K je tuhost pružiny a g je gravitační konstanta. Blokové schéma a průběh simulace lze vidět na obrázku 5.10.



Obrázek 5.9: Schéma a výsledky modelu Van der Polova oscilátoru.



Obrázek 5.10: Blokové schéma a výsledky simulace systému závaží na pružině.

Kapitola 6

Závěr

Cílem této práce bylo navrhnout a implementovat aplikaci obsahující editor a simulátor stochastických časovaných Petriho sítí a spojitých blokových schémat s využitím technologie WebAssembly. Grafický editor byl vytvořen pomocí jazyku Typescript s využitím knihovny React pro grafické uživatelské rozhraní. Simulátor byl implementován v jazyce C++ a pomocí nástroje Emscripten kompilován do WebAssembly modulu. Výsledná webová aplikace byla nasazena pomocí *GitHub Pages*, díky čemuž je volně dostupná bez nutnosti jakékoliv instalace. Aplikace poskytuje uživateli jednoduché grafické rozhraní, ve kterém může vytvořit model pomocí zmíněných dvou formalismů, provést simulaci nad vytvořeným modelem a získané výsledky si nechat zobrazit pomocí grafů. Pro účely podpory výuky byla implementována sada 10 příkladů, díky kterým se uživatel lépe seznámí s danou problematikou a ovládáním aplikace. Příklady je taktéž možné využít jako vstup pro tvorbu vlastního modelu.

Mezi možnosti dalšího rozvoje patří rozšíření o podporu dalších typů spojitých bloků, možnost výběru použité numerické metody, podporu inhibičních hran a pravděpodobnostních přechodů pro Petriho sítě, doplnění algoritmů pro lepší validaci modelů nebo přidání zcela nové oblasti jako např. podpora pro modelování a simulaci systémů hromadné obsluhy. Při vývoji byl kladen důraz na maximální rozšiřitelnost a za tímto účelem byla vytvořena celá řada prostředků, které lze při implementaci dalších rozšíření využít, čímž by mělo dojít k výrazné časové úspoře. Pro zlepšení práce s aplikací se nabízí přidat možnost uložení a nahrání vytvořeného modelu a export získaných statistik.

Literatura

[1] *About CMake*. [cit. 2022-04-26]. Dostupné z: https://cmake.org/overview/.

[2] *JavaScript with syntax for types*. [cit. 2022-5-1]. Dostupné z: https://www.typescriptlang.org/.

[3] *React – a JavaScript library for building user interfaces*. [cit. 2022-5-1]. Dostupné z: https://reactjs.org/.

[4] *Sass - CSS preprocesor*. [cit. 2022-5-1]. Dostupné z: https://www.interway.sk/blog/sass-css-preprocesor.html.

[5] *Stack overflow developer survey 2021*. [cit. 2022-04-19]. Dostupné z: https://insights.stackoverflow.com/survey/2021.

[6] *A tool for editing, simulating, and analyzing colored Petri nets*. [cit. 2022-04-07]. Dostupné z: http://cpntools.org/.

[7] *Wasmtime*. Dostupné z: https://docs.wasmtime.dev/.

[8] *Getting started with Redux*. Dec 2021 [cit. 2022-04-18]. Dostupné z: https://redux.js.org/introduction/getting-started.

[9] *Modelica®Language specification version 3.5*. Modelica Association, Feb 2021 [cit. 2022-04-17]. Dostupné z: https://modelica.org/documents/MLS.pdf.

[10] Arushanyan, R. *Redux-cool philosophy*. DEV Community, Apr 2021 [cit. 2022-04-28]. Dostupné z: https://dev.to/redux-cool/redux-cool-philosophy-18jp.

[11] Clark, L. *Standardizing WASI: A system interface to run WebAssembly outside the web*. Mar 2019 [cit. 2022-04-29]. Dostupné z: https://hacks.mozilla.org/2019/03/standardizing-wasi-a-webassembly-system-interface/.

[12] Daubenschütz, T. *WASM synth, or, how music taught me the beauty of math*. Feb 2020 [cit. 2022-04-20]. Dostupné z: https://timdaub.github.io/2020/02/19/wasm-synth/.

[13] Desel, J. a Juhás, G. *What Is a Petri Net? - Informal Answers for the Informed Reader*. 2001. ISBN 978-3-540-45541-7. Dostupné z: https://link.springer.com/chapter/10.1007/3-540-45541-8_1.

[14] Duggal, N. *Uses of C++: What is C++ used for: C++ applications: Simplilearn*. Simplilearn, Dec 2021 [cit. 2022-04-19]. Dostupné z: https://www.simplilearn.com/tutorials/cpp-tutorial/top-uses-of-c-plus-plus-programming.

[15] Eberhardt, C. *What is webassembly?* O'Reilly Media, Inc. Dostupné z: https://www.oreilly.com/library/view/what-is-webassembly/9781492076902/ch04.html.

[16] *EMSCRIPTEN documentation.* 2015 [cit. 2022-04-16]. Dostupné z: https://emscripten.org/docs/index.html.

[17] Fritzson, P., Pop, A., Abdelhak, K., Ashgar, A., Bachmann, B. et al. The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development. *Modeling, Identification and Control.* Norwegian Society of Automatic Control. 2020, sv. 41, č. 4, s. 241–295. DOI: 10.4173/mic.2020.4.1.

[18] Haque, S. *React.js with factory pattern : building complex UI with ease.* Bits and Pieces, Sep 2020 [cit. 2022-05-1]. Dostupné z: https://blog.bitsrc.io/react-js-with-factory-pattern-building-complex-ui-with-ease-fe6db29ab1c1.

[19] Hrubý, M. *Modelování a simulace - spojité modelování - Slajdy pro předmět IMS.* Dec 2005 [cit. 2022-05-1]. Dostupné z: http://perchta.fit.vutbr.cz/vyuka-ims/uploads/1/spoja.pdf.

[20] *MATLAB & SIMULINK.* Humusoft [cit. 2022-04-16]. Dostupné z: https://www.humusoft.cz/matlab/simulink/.

[21] Jensen, K. a Kristensen, L. M. *Coloured Petri nets modelling and validation of Concurrent Systems.* Springer Berlin, 2014.

[22] *Webassembly.* [cit. 2022-04-02]. Dostupné z: https://developer.mozilla.org/en-US/docs/WebAssembly.

[23] Peringer, P. *Modelování a simulace* [online]. 2021 [cit. 2022-04-22]. Dostupné z: https://wis.fit.vutbr.cz/FIT/st/cfs.php.cs?file=%2Fcourse%2FIMS-IT%2Ftexts%2Fopora-ims.pdf&cid=14664.

[24] Petri, C. *Kommunikation mit Automaten.* Rheinisch-Westfälisches Institut f. instrumentelle Mathematik an d. Univ., 1962. Schriften des Rheinisch-Westfälischen Institutes für Instrumentelle Mathematik an der Universität Bonn. Dostupné z: https://books.google.cz/books?id=NCZMvAEACAAJ.

[25] *Webassembly.* [cit. 2022-04-02]. Dostupné z: https://webassembly.org/.

[26] *Xcos for very beginners.* Scilab Enterpripses, 2013 [cit. 2022-04-16]. Dostupné z: https://www.scilab.org/sites/default/files/Xcos_beginners.pdf.