VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

ANALÝZA PLÁNOVATELNOSTI ÚLOH REÁLNÉHO ČASU S OHLEDEM NA NEJISTOTU

SCHEDULABILITY ANALYSIS OF REAL-TIME TASKS UNDER UNCERTAINTY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **SAMUEL ČUS**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JOSEF STRNADEL, Ph.D.**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

150944

Ústav: Ústav počítačových systémů (UPSY)
Student: **Čus Samuel**
Program: Informační technologie
Název: **Analýza plánovatelnosti úloh reálného času s ohledem na nejistotu**
Kategorie: Modelování a simulace
Akademický rok: 2023/24

Zadání:

1. Proveďte rešerši v oblasti modelování a analýzy vlastností systémů reálného času; obzvláště se zabývejte problematikou analýzy plánovatelnosti úloh reálného času.
2. Identifikujte nejistoty související se systémy a úlohami reálného času. Vytvořte přehled metod a nástrojů pro analýzu plánovatelnosti úloh reálného času a přístupů k řešení problému analýzy plánovatelnosti s ohledem na nejistotu.
3. Shrňte klíčové pojmy a principy související s technikou statistického ověřování modelů (angl. Statistical Model Checking, SMC). Identifikujte prostředky SMC vhodné k modelování množin úloh reálného času a analýze jejich plánovatelnosti s ohledem na nejistotu; proveďte rešerši v této oblasti.
4. Navrhněte kroky procesu analýzy plánovatelnosti úloh reálného času s využitím SMC. Diskutujte množiny úloh reálného času a scénáře nejistot vhodné pro ověření použitelnosti navrženého procesu a jeho zhodnocení.
5. Implementujte vlastní, popř. dostatečně rozšiřte existující, přístup k modelování a analýze plánovatelnosti úloh reálného času s ohledem na nejistotu.
6. Připravte vhodné sady úloh reálného času s cílem ověřit jejich plánovatelnost pomocí přístupu z bodu 5 při zohlednění různých zdrojů nejistot.
7. Zhodnoťte použitý přístup a výsledky pomocí něj dosažené; identifikujte silné a slabé stránky přístupu, navrhněte možné směry jeho vylepšení a rozveďte ty, které považujete za nejvíce perspektivní.

Literatura:

- David, A., Larsen, K., Legay, A., Mikučionis, M., Poulsen, D.: Uppaal SMC tutorial. International Journal on Software Tools for Technology Transfer. Springer Berlin Heidelberg, 2015, pp. 397 - 415, Vol. 17, No. 4. ISSN 1433-2779, DOI 10.1007/s10009-014-0361-y.
- Shan, L., Graf, S., Quinton, S., Fejoz, L.: A Framework for Evaluating Schedulability Analysis Tools. In: Models, Algorithms, Logics and Tools, 2017, 21 p. LNCS, Vol. 10460. Springer, Cham. DOI 10.1007/978-3-319-63121-9_27.
- Laplante, P., Ovaska, S.:. Real-Time Systems Design and Analysis: Tools for the Practitioner, 2011, 584 p. DOI 10.1002/9781118136607.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3 zadání.
- Představení koncepce modelu úloh reálného času a procesu analýzy plánovatelnosti úloh reálného času s ohledem na nejistotu.

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

Vedoucí práce: **Strnadel Josef, Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1.11.2023
Termín pro odevzdání: 9.5.2024

Datum schválení: 30.10.2023

Abstrakt

Cílem této bakalářské práce je seznámení se se systémy pracujícími v reálném čase, nejistotami a plánovacími mechanismy, souvisejícími s těmito systémy, statistickým ověřováním modelů, dále je to návrh a implementace přístupu k analýze plánovatelnosti s ohledem na nejistoty, vytvoření vhodných sad úloh reálného času a ověření jejich plánovatelnosti. Zaměřil jsem se na nejistoty způsobené přerušením systému a nedeterministickými parametry úloh. Zadaný problém jsem řešil vytvořením sad úloh, zavedením nejistot do systému a analýzou plánovatelnosti úloh. Modelování systému a jeho analýza bylo provedeno v nástroji UPPAAL SMC pro porovnaní také v nástroji Cheddar u vybraných sad úloh.

Abstract

The goal of this bachelor's thesis is to familiarize with real-time systems, uncertainties and scheduling mechanisms related to these systems, statistical model checking, as well as the design and implementation of an approach to schedulability analysis under uncertainties, creating suitable sets of real-time tasks, and verifying their schedulability using the approach. I have mainly incorporated uncertainties caused by system interruptions and undeterministic task parameters. To address the given problem, I created a set of tasks, incorporated uncertainties into the system, and analyzed the schedulability of the tasks. The system modeling and analysis were conducted using the UPPAAL SMC tool, and, for comparison, also using the Cheddar tool with selected sets of tasks.

Klíčová slova

reálný čas, systémy reálného času, nejistoty, úlohy reálného času, analýza plánovatelnosti, UPPAAL, statické ověřování modelů, ověřování modelů, plánovací mechanismy, časované automaty, modální logika, verifikace systémů

Keywords

real-time, real-time systems, uncertainties, real-time tasks, schedulability analysis, UPPAAL, statistical model checking, model checking, scheduling mechanisms, timed automatons, modal logic, system verification

Citace

ČUS, Samuel. *Analýza plánovatelnosti úloh reálného času s ohledem na nejistotu.* Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Josef Strnadel, Ph.D.

Analýza plánovatelnosti úloh reálného času s ohledem na nejistotu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Josefa Strnadela, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Samuel Čus
9. května 2024

Poděkování

Velmi rád bych poděkoval vedoucímu práce panu Ing. Josefu Strnadelovi, Ph.D za cenné a užitečné rady při tvorbě této bakalářské práce.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Systémy reálného času (dále už jen RT systémy) jsou neodmyslitelnou součástí našeho každodenního života. Jsou klíčové v oblasti zdravotnictví, energetiky a automobilové výroby, kde se tyto systémy využívají pro zajištění spolehlivosti, efektivity a především bezpečnosti. V těchto oblastech je vyžadována vysoká preciznost a načasování. Z těchto důvodů musí systémy zajistit zpracování dat v reálném čase. V kontextu chytrých zařízení, elektronických spotřebičů hraje využití těchto systémů významnou roli v otázkách týkajících se optimalizace provozu, synchronizace časových údajů a celkově zajištění komfortu uživatelů těchto zařízení.

Vzhledem k očekávanému nárůstu počtu systémů reálného času v blízké budoucnosti je proto potřeba věnovat pozornost také analýze plánovatelnosti, abychom mohli zajistit dodržení časových mezí.

Cílem této bakalářské práce je seznámení se s systémy pracujícími v reálném čase, nejistotami a plánovacími mechanismy souvisejícími s těmito systémy, statistickým ověřováním modelů, návrh a implementace přístupu k analýze plánovatelnosti s ohledem na nejistoty, vytvoření vhodných sad úloh reálného času a ověření jejich plánovatelnosti

Implementace modelu s využitím statistického ověřování modelů je provedena v nástroji UPPAAL SMC. Pro kontrolu a porovnání výsledků experimentů byl využit nástroj Cheddar. Nejistoty byly do systému zaneseny prostřednictvím přerušení, které narušují zpracování úloh. Dále je možné u úloh zvolit nedeterministické parametry.

Kapitola 2

Teoretická část

V této kapitole jsou na začátku definovány základní pojmy spojené s RT systémy. Následně jsou detailněji popsány RT systémy a jejich klasifikace. Dále si popíšeme úlohy reálného času (dále už jen RT úlohy), plánovací mechanismy používané pro RT systémy a čím jsou všechny tyto oblasti specifické v porovnání se systémy, kde se nás časové omezení nezajímají. Na závěr této kapitoly se budeme věnovat nejistotám v RT systémech.

2.1 Základní pojmy

Reálný čas

Reálný čas v kontextu RT systémů značí, že je požadována správná odezva „včas“. Je potřeba zmínit fakt, že odezva „včas“ závisí na konkrétní situaci. Nemusí proto vždy platit, že odezva v reálném čase = okamžitá odezva. [18]

Plánování

Plánování je rozhodovací proces využívaný v mnoha odvětvích. Zabývá se přidělováním zdrojů úlohám v daném čase a jeho cílem je optimalizovat jeden nebo více cílů.

Zdroje a úlohy mohou mít mnoho forem v závislosti na konkrétní organizaci. Zdroje mohou být například stroje v dílnách, ranveje na letišti, pracovníci na stavbě a úlohy, které souvisí s těmito zdroji jsou pak operace ve výrobním procesu, přistání a odlety na letišti, jednotlivé fáze ve stavebním projektu nebo vykonávání počítačových programů. Každá úloha může mít určitou úroveň priority, čas začátku vykonávání nebo čas, do kterého se musí úloha dokončit.

Plánování hraje důležitou roli ve většině výrobních a produkčních systémů stejně jako ve většině prostředí zpracování informací. Informace o tomto pojmu byly převzaty z [15].

Analýza plánovatelnosti

Analýza plánovatelnosti (anglicky schedulability analysis) představuje metodu pro zhodnocení časové korektnosti RT systémů. Provádí se off-line, což znamená, že se analýza provádí mimo běh systému. Cílem je posoudit, zda všechny úlohy splňují své časové meze (anglicky deadlines) během reálného provozu. [16]

V praxi to znamená, že analyzujeme, zda jsou všechny úlohy v plánu schopny dokončit své vykonávání v předepsaných termínech. Tímto způsobem můžeme posoudit, zda je systém schopen zajistit, aby všechny úlohy byly provedeny včas.

Nejistota

Nejistotu lze popsat jako stav, kdy nemáme dostatek znalostí nebo také jako nedostatek jistoty. Není možné přesně popsat aktuální stav, budoucí výsledek nebo více než jeden možný výsledek. [8]

2.2 Systémy pracující v reálném čase

U RT systémů požadujeme nejen správnou reakci na vstup, ale také dodržení vymezené časové meze. Nedodržením časové meze není myšleno pouze dokončení po termínu, ale za nedodržení mezí se považuje i situace, kdy se úloha dokončí příliš brzy. Požadavky na jednotlivé systémy se liší, a to jak z již zmíněného důsledku nedodržení časových mezí, tak i z hlediska bezpečnosti a provozuschopnosti. [9]
RT systémy zařazujeme podle typů mezí odezev do jedné z následujících kategorií:

- tvrdé (anglicky hard)
- měkké (anglicky soft)
- pevné (anglicky firm)

Každá kategorie se liší podle typů mezí odezev a tím, jaká je reakce na nedodržení časových mezí a jestli se dokáže systém zotavit při jejich nedodržení. Podrobnější popis těchto mezí je popsán v tabulce 2.1.

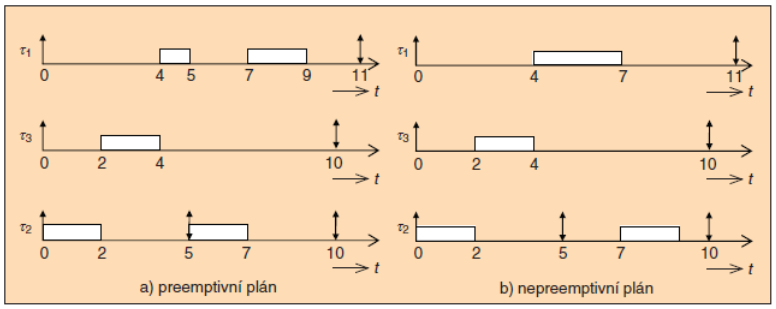
Meze odezvy	**Čas. kritičnost**	**Charakteristika**
měkká	malá	Meze odezev tohoto typu jsou optimální k dosažení uživatelem očekávané kvality služeb poskytovaných systémem RT. Nedodržení mezí odezev zpravidla vede k dočasnému poklesu kvality služeb se zanedbatelnými dopady na okolí systému.
tvrdá	velká	Každá mez odezev tohoto typu musí být bezpodmínečně do držena nedodržení jediné z nich vede k nevratným a trvalým následkům v okolí systému, neboť reakce v okolí systému nebyla provedena včas. Pokusy o dodržení jakýchkoliv dalších mezí jsou nemožné nebo zbytečné, jelikož vlivem selhání podstatné vlastnosti systému již neexistuje způsob vedoucí ke zmírnění vzniklých následků.
pevná	střední	Meze odezev tohoto typu se typicky vyznačují předem danou tolerancí jejich nedodržení překročení této tolerance může mít vážný dopad na okolí systému. Pokud je tolerance navrhována s dostatečnou rezervou, systém může předvídat její potenciální překročení s předstihem dostačujícím ke včasnému spuštění zotavovacích mechanismů. Při překročení to lerance a neschopnosti systému se zotavit se ze vzniklé situace může být například signalizována nutnost řešit vzniklou situaci prostředky mimo systém. Následky nedodržení mezí tohoto typu jsou typicky vratné a mohou být dostatečně zmírněny či zcela odstraněny, neselžou-li záchranné mechanismy.

Tabulka 2.1: Klasifikace a charakteristika typů mezí odezev RT systémů.
Informace použité v tabulce byly převzaty z [20]

2.2.1 Plánovač

Plánovač je nástroj, jehož úkolem je přidělovat jednotlivým úlohám procesorový čas. Většina plánovačů je v dnešní době prioritních. Prioritní plánovače přidělují úlohám procesorový čas na základě jejich priority. V ideálním případě by měla být bežící úloha vždy ta s největší prioritou. [18]
Podle schopnosti přerušit běžící úlohu a přepnout ji na jinou rozdělujeme plánovače do dvou kategorií:

- preemptivní – zajišťují přidělení zdrojů úloze s nejvyšší prioritou přerušením běžící úlohy
- nepreemptivní – přepnutí kontextu je provedeno až po dokončení běžící úlohy



Obrázek 2.1: **Porovnání preemptivního a nepreemptivního plánovače** [18]

2.2.2 Plán

Plán určuje, jak budou jednotlivé úlohy prováděny v čase. Znázorňuje se především pomocí časových diagramů. V případě, že plán znázorňuje úlohy, u kterých nedošlo k porušení časových mezí, nazýváme plán přípustným. Pokud existuje pro množinu úloh přípustný plán, můžeme říci, že tato množina úloh je plánovatelná. Postup, který určuje, zda je plán pro danou množinu přípustný se, nazývá test plánovatelnosti. [18]

2.2.3 RT úloha

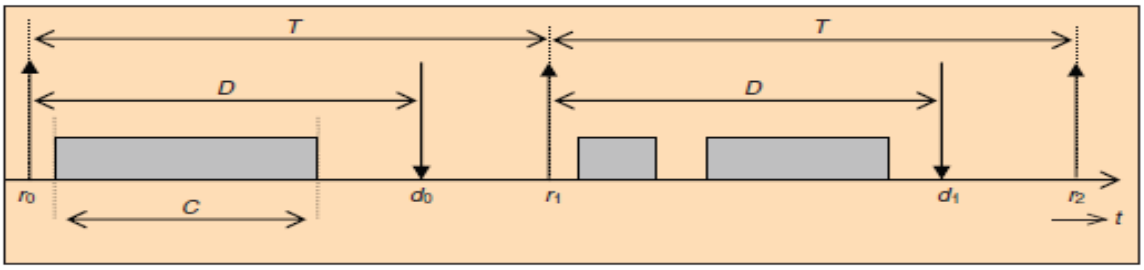
RT úloha je operace, která je naplánována, a počítačový systém ji musí vykonat v reálném čase. Lze si ji představit jako výpočetní jednotku provádějící konkrétní funkci. Množina úloh poté obsahuje všechny úlohy daného systému. Roli v rámci plánování hraje především priorita. Úlohy s vyšší prioritou mají přednost před úlohami s nižší prioritou. Pokud byla priorita úloze přiřazena již před zavedením do systému, nazýváme ji statická. Některé plánovací mechanismy ale pracují i s dynamickou prioritou, která se mění v čase.
Podle toho, jak můžeme popsat časy mezi příchody požadavků na zahájení úloh, rozdělujeme úlohy do jedné z následujících kategorií:

- periodické: požadavky přichází s periodou T
- aperiodické: interval mezi příchody nelze určit
- sporadické: známe pouze nejkratší možnou dobu mezi příchody požadavků

Pro znázornění úlohy používáme model úlohy. Ten může obsahovat jak základní, tak i dynamické parametry. Čím více parametrů bude model úlohy obsahovat, tím více se bude model blížit skutečnému systému. Spolu s počtem parametrů ale rostou nároky kladené na plánovač i nároky na analýzu parametrů systému. Základní parametry modelu úlohy jsou popsány a znázorněny v tabulce 2.2 a obrázku 2.2. Informace použité v této sekci byly čerpány z [18].

Parametr	**Symbol**
absolutní čas vyvolání úlohy	r
perioda	T
nejdelší doba běhu úlohy	C
relativní časová mez odezvy úlohy	D
absolutní časová mez odezvy úlohy	d
priorita úlohy	P

Tabulka 2.2: Základní parametry RT úlohy

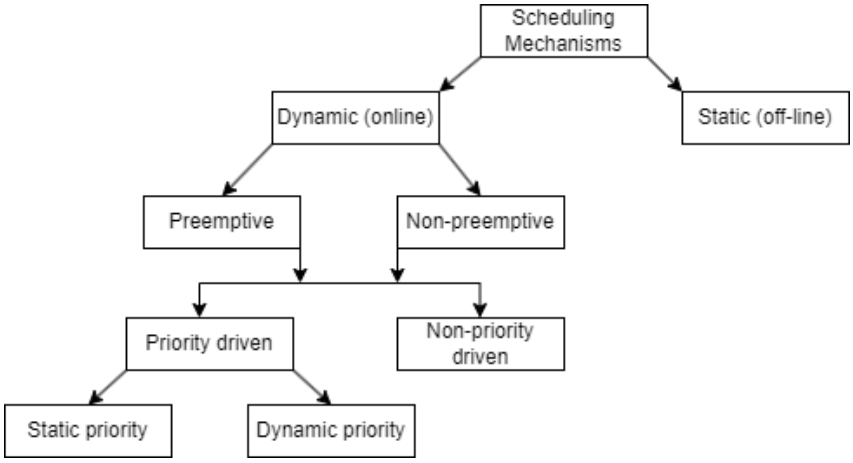


Obrázek 2.2: **Ilustrace základních parametrů úloh** [18]

2.3 Plánovací mechanismy

V této sekci si popíšeme plánovací mechanismy, které se používají pro plánování v systémech RT. Plánovací mechanismus určuje, jakým způsobem jsou úlohám přiřazovány priority. Na základě plánovacího mechanismu pak plánovač určuje pořadí úloh.

Podle toho, kdy dochází k přiřazení priorit úlohám, rozdělujeme plánovací mechanismy na off-line a on-line. Off-line plánovací mechanismy jsou charakteristické tím, že přiřazují úlohám prioritu ještě před začátkem běhu systému. Obecně jsou jednodušší než on-line plánovací mechanismy, které vytváří plán až za běhu systému a jsou schopné reagovat na změny. On-line plánovací mechanismy dále rozdělujeme podle schopnosti přerušit běžící úlohu na preemptivní a nepreemptivní. Ty dále dělíme na priortní a neprioritní. Posledním rozdělením, které zmíníme, je rozdělení prioritních mechanismů podle způsobu přiřazování priorit. Rozdělujeme je na statické a dynamické. Pokud není uvedeno jinak, informace v této sekci byly čerpány z [9] [19].



Obrázek 2.3: **Rozdělení plánovacích mechanismů**

Mechanismy statického přiřazování priority

Statické plánování je jednoduché a předvídatelné, protože nedochází ke změně priority za běhu systému. Je vhodné především pro nezávislé periodické úlohy.

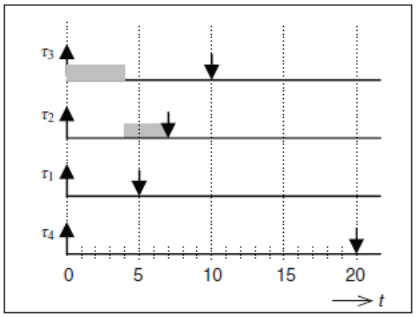
Úloha	**r**	**C**	**D**	**T**
τ_1	0	3	5	20
τ_2	0	3	7	12
τ_3	0	4	10	10
τ_4	0	3	20	20

Tabulka 2.3: Úlohy a jejich parametry použité pro ilustraci RMA a DMA mechanismů

RMA

RMA (*Rate monotonic assignment*) je plánovací mechanismus, jehož hlavní myšlenkou je přiřadit statické priority úlohám na základě jejich period, přičemž čím kratší je perioda, tím vyšší je priorita úlohy. Jinak řečeno, čím je úloha volaná častěji, tím má vyšší prioritu. Pokud RMA mechanismus není schopen zajistit plánovatelnost množiny úloh, existují dva způsoby, jak na tuto situaci reagovat. Prvním způsobem je změna parametrů RT úloh, kvůli čemuž by ale došlo k odklonu od verifikované specifikace RT systému, což by mohlo způsobit neočekávané chování. Další variantou je použití jiného plánovacího mechanismu. Pokud množina úloh není plánovatelná pomocí RMA, je to často způsobeno tím, že RMA mechanismus je optimální pouze pro úlohy, ve kterých se rovnají jejich parametry D a T. Podle RMA platí nutná a postačující podmínka pro to, aby byla sada úloh plánovatelná:

$$\sum_{i=1}^{n} \frac{C_i}{D_i} \leq n(2^{1-n} - 1)$$

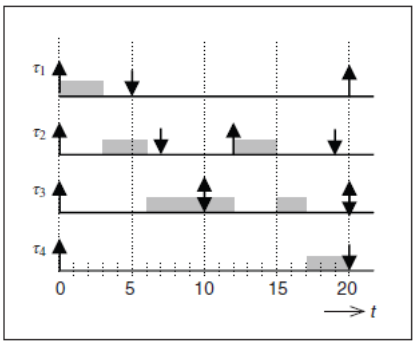


Obrázek 2.4: **RMA mechanismus** [5]

DMA

DMA (*Deadline monotonic assignment*) představuje mechanismus statického přiřazování priorit. Základem mechanismu je předpoklad, že čím menší je hodnota časové meze, tím vyšší je její priorita. Z toho vyplývá, že priorita úlohy je nepřímo úměrná parametru D úlohy. Použití mechanismu DMA je vhodné pro množinu úloh, kde pro jejich parametry D a T platí, že $D \leq T$. Můžeme tedy říci, že mechanismus DMA lze použít pro plánování těch množin úloh, které nejsou plánovatelné pomocí mechanismu RMA.
Podle DMA platí nutná a postačující podmínka pro to, aby byla sada úloh plánovatelná:

$$\sum_{i=1}^{n} \frac{C_i}{D_i} \leq n(2^{1/n} - 1)$$



Obrázek 2.5: **DMA mechanismus** [5]

Mechanismy dynamického přiřazování priority

Dynamické plánování umožňuje změnit prioritu úlohám během provádění na základě dynamických parametrů, jako je například relativní volnost. Tento způsob plánování dokáže dobře reagovat na změny, ale oproti statickému přiřazování priorit je složité a nepředvídatelné. Jejich použití je vhodné pro úlohy aperiodické či sporadické, závislé úlohy nebo také pro úlohy plánované při přetížení systému.

Úloha	r	C	D	T
τ_1	0	2	5	5
τ_2	0	4	7	7

Tabulka 2.4: Úlohy a jejich parametry použité pro ilustraci EDF a LLF mechanismů

EDF

EDF (*Earliest deadline first*) je dynamický mechanismus, který přiřazuje v čase t nejvyšší prioritu těm úlohám, kterým zbývá do uplynutí časové meze méně času. Parametr RT úlohy reprezentující tuto hodnotu značíme $D(t)$.

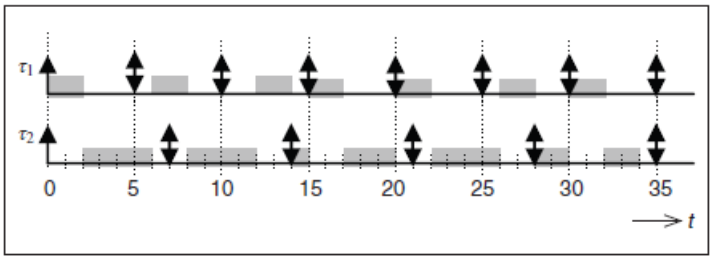
EDF je optimální ve smyslu proveditelnosti: pokud existuje proveditelný plán pro danou sadu úloh, pak je ho mechanismus schopen nalézt. Mechanismus EDF nedělá žádný předpoklad o periodicitě úloh, a proto může být použit pro plánování jak periodických, tak aperiodických úloh. [5]

Množinu periodických úloh s termíny rovnajícími se periodám lze naplánovat pomocí algoritmu EDF tehdy, když faktor využití procesoru je menší nebo roven 1:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

Hybridní množinu úloh lze naplánovat pomocí EDF mechanismu, pokud je splněna podmínka:

$$\sum_{i=1}^{n} \frac{C_i}{D_i} \leq 1$$

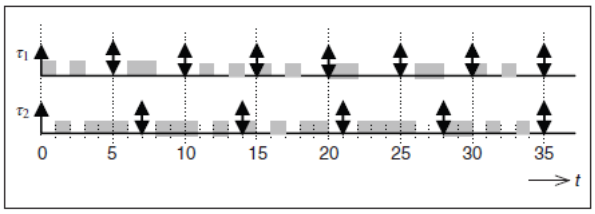


Obrázek 2.6: **EDF mechanismus** [5]

LLF

LLF (*Least laxity first*) přiřazuje prioritu úlohám podle hodnoty parametru $L(t)$, který určuje relativní volnost. Relativní volnost vyjadřujeme jako rozdíl parametrů $D(t)$ a $C(t)$ a mohli bychom ji popsat jako dobu, po kterou se může odložit provádění úlohy tak, aby se úloha nepřekročila časovou mez. Tento mechanismus je optimální a plánovatelnost sady úloh lze zaručit pomocí EDF testu plánovatelnosti.

Pomocí LLF je možné zamezit překročení časových mezí dříve než pomocí mechanismu EDF. Na stejně velkém časovém úseku dochází k přehodnocování priorit u LLF častěji než při použití EDF mechanismu. Díky tomu dokáže LLF dříve předpovídat překročení časové meze. Když je úloze přidělen procesorový čas, relativní volnost úlohy zůstává konstantní, zatímco relativní volnost připravených úloh postupně klesá. Informace o LLF byly převzaty z [5].



Obrázek 2.7: **LLF mechanismus** [5]

Alternativní plánovací mechanismy

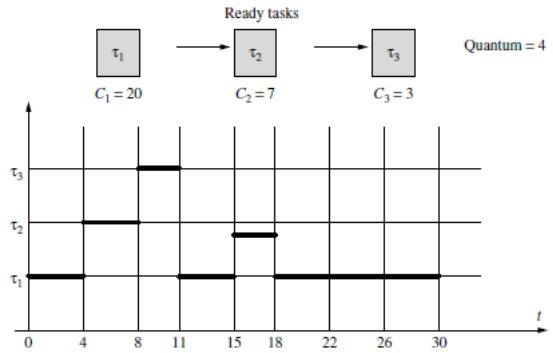
FPS

FPS (*Fixed priority scheduling*) je jednoduchý plánovací mechanismus. Úlohám je předem přidělena priorita, která je neměnná. Úlohy s vyšší prioritou mají přednost před úlohami s prioritou nižší.

Round robin

Round robin představuje plánovací mechanismus, který přiděluje procesor postupně každé připravené úloze v rámci definovaného časového kvanta. Jak se tento mechanismus bude chovat, závisí na velikosti kvanta. Čím vyšší je hodnota časového kvanta, tím se zvyšuje doba odezvy, zatímco zmenšování kvanta zvyšuje přepínání úloh a režie s tím spojená není zanedbatelná.

Pokud dojde k dokončení úlohy dříve, než nastane překročení časové meze, uvolní se procesor a přidělí se další připravené úloze. V opačném případě, pokud se úloha nestihne dokončit do konce kvanta, je přerušena a vloží se na konec fronty připravených úloh. Informace o mechanismu byly použity z [5].



Obrázek 2.8: **Příklad plánování pomocí mechanismu Round robin** [5]

FIFO

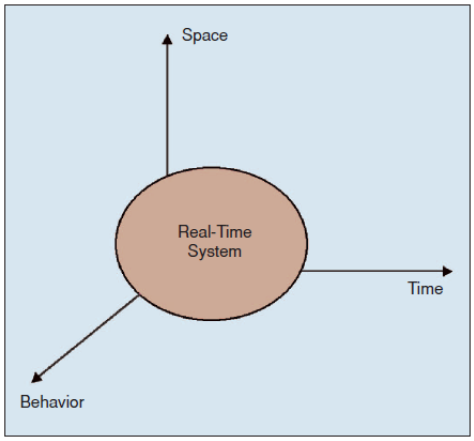
FIFO (*First in first out*) patří mezi nejjednodušší plánovací mechanismy. Přiděluje procesor úlohám v pořadí, v jakém dorazily do systému, a jedná se o nepreemptivní mechanismus. Znamená to tedy, že procesor je přidělen té úloze, která je v systému nejdéle. Často dochází k situacím, kdy jsou kvůli tomu ovlivněny krátké úlohy, pokud se před nimi ve frontě nachází úlohy s delším výpočetním časem. [5]

2.4 Nejistoty

Identifikace, řízení a odstranění nejistot jsou nezbytnou součástí konstruování komplexních systémů. V RT systémech nejistoty existují na různých úrovních a ve více dimenzích, což vyžaduje pečlivý a komplexní přístup k jejich zvládání. Pokud není řečeno jinak, informace v této sekci byly čerpány z [10].

2.4.1 Dimenze nejistot

V RT systémech se nejistoty vyskytují ve třech dimenzích: čas, prostor a chování (viz 2.9). Nemůžeme přímo aplikovat Heisenbergův princip neurčitosti, který je matematická vlastnost dvou kanonicky konjugovaných veličin a říká, že čím přesněji určíme jednu z konjugovaných vlastností, tím méně přesně můžeme určit tu druhou. V našem případě by to bylo možné vysvětlit takto: pokud se pokusíme snížit nejistotu v jedné z dimenzí, nejistota v ostatních dvou dimenzích naroste.



Obrázek 2.9: **Dimenze nejistot RT systémů** [10]

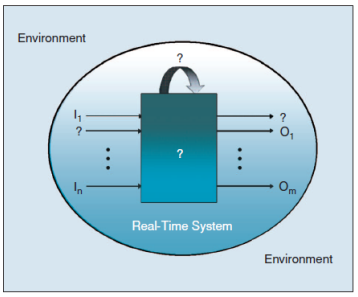
2.4.2 Zdroje nejistot

Pokud se podíváme na RT systém na obrázku 2.9 a budeme hledat potenciální zdroje nejistot, a to především v dimenzi chování, musíme si RT systém představit jako transformaci množiny vstupů a aktuálního stavu do nového stavu a množiny výstupů. Každý z těchto prvků může být zdrojem nejistoty.

Následující podsekce pronikají hlouběji do příčin nejistot v prostředí, vstupech, výstupech a chování.

Prostředí

Nejistota prostředí může pramenit z mnoha různých zdrojů. Může to být způsobeno například chaotickým systémem.



Obrázek 2.10: **Zdroje nejistot RT systému** [10]

Chaotické systémy poznáme podle toho, že i malá změna ve vstupech vede k radikálním změnám v chování systému a v jeho výstupech. S tím souvisí i fakt, že poškozený výstup RT systému, který je vstupem pro stabilní systém pod kontrolou může způsobit, že tento systém se může jevit jako chaotický, i když chaotický není.

Dalším forma nejistoty prostředí vychází z nejednoznačně definovaných, neúplných nebo nekonzistentních softwarových požadavků. V případě, že těmto nejistotám není věnována dostatečná pozornost, povede to k mnohem zákeřnějším nejistotám, které budou velmi těžko odhalitelné.

Další forma nejistoty souvisí s prostředím, ve kterém systém operuje. Může se jednat například o vliv přírodních živlů na systém. Fyzické prostředí ovlivnit nemůžeme, ale systém by měl být navržen a zkonstruován tak, aby dokázal těmto vlivům vzdorovat.

Poslední formou nejistoty prostředí, kterou zmíníme je testování. Testování využíváme pro odhalení případných zdrojů nejistot. Musíme si uvědomit, že testování samotné představuje nejistotu. Nedokážeme s jistotou říct, jestli je pokrytí testů dostatečné a nebo jestli používáme vhodnou testovací strategii. Fakt, že systém prošel všemi testy nezaručuje, že žádné chyby nebo nejistoty neobsahuje.

Vstupy

Nejistoty související se vstupy RT systémů vznikají především kvůli špatně fungujícím zařízením, šumu nebo konverzí signálů. Některé špatné vstupy ale mohou být následkem problémů souvisejících s prostředím, ve kterém systém pracuje. Vstupy musí být verifikovány, filtrovány a musí být ověřena jejich přiměřenost před jejich použitím.

Výstupy

Výstupy RT systému mohou být ovlivněny poruchou zařízení i chybami při převodu, což představuje nejistotu pro systém pod kontrolou. Mohlo by tak dojít k situaci, kdy systém dostává znehodnocené vstupy od systému pod kontrolou na základě svých vlastních znehodnocených výstupů.

Chování

Nejisté chování je třída nejistoty, která zahrnuje problémy související s časováním, plánováním, nejistotou chování komponent a nejistotou programovacího jazyka.

Nejistota chování komponent v běžně dostupném hardwaru a softwaru je problém, který nesmíme opomenout. Existují různé techniky, pomocí kterých můžeme nejistotu snížit. Například lze použít fault-injection, pokud je k dispozici zdrojový kód a nebo se využívá rigorózní black-box testování komponent. Další forma této nejistoty je způsobena použitím programovacích jazyků. V objektově orientovaných jazycích se preferuje použití kompozice před dědičností. V objektově orientovaném a procedurálním programování mohou neomezené rekurze, mrtvý a nedosažitelný kód, nekonečné smyčky a neodstraněné ladicí příkazy vést k nejistému chování.

2.4.3 Přerušení

Jako významný zdroj nejistoty působí v systémech RT také přerušení. Přerušení signalizuje, že došlo k události, na kterou je potřeba zareagovat. Příkladem může být informace o tom, že došlo k stisknutí klávesy, nebo může jít o pravidelný hodinový signál.

Po příchodu přerušení se přeruší provádění hlavního programu, dojde k zákazu dalších přerušení, přerušení se obslouží, pak se opět přerušení povolí a pokračuje se v provádění hlavního programu. Informace o přerušeních byly převzaty z [11].

Maskovatelná přerušení

Procesory mají dva typy přerušení - maskovatelné a nemaskovatelné. Maskovatelné přerušení se liší od nemaskovatelného tím, že programátor může jeho provádění zakázat pomocí speciální instrukce. K obsluze přerušení tedy nedojde.

Typy nejistot	**Příznaky**	**Možné příčiny**	**Možná řešení**
Systém pod kontrolou	Neobvyklé vstupy Neobvyklé výstupy	Povaha aplikace Vadný hardware	Použít agresivní design odolný proti chybám
Provozní prostředí	Neobvyklé vstupy Neobvyklé výstupy	písek, sůl, náraz, přírodní vlivy	Použít agresivní design odolný proti chybám
Požadavky	Nedostatek požadavků Většina požadavků nespecif.	Neúplné požadavky Nekonzistentní požadavky	Požadavky na základě cíle Analýza
Testování	Systém prochází testy Při nasazení selže	Nedostatečné pokrytí Špatný testovací režim	Zlepšení test. procesu
Vstup	Neočekávané chování Vysvětlující komentáře	Nestabilní zdroje vstupů Nekvalitní hardware	Kalmanovy filtry, fúze dat mediánové filtry
Výstup	Neočekávané chování Vysvětlující komentáře	Nestabilní zdroje vstupů Vadný vstup od systému pod kontrolou	Kalmanovy filtry, fúze dat mediánové filtry
Stav	Neočekávané chování	Skoky v kódu	Ověřování modelu
Jazyk	Vysvětlující komentáře	Chyby kompilátoru	Otestování překladače Zlepšení program. technik

Tabulka 2.5: Přehled druhů nejistot v RT systémech, jejich znaky, jejich možné zdroje a možná řešení problému. Informace použité v tabulce byly převzaty z [10]

2.4.4 Přístupy zahrnující nejistotu

Elastic model

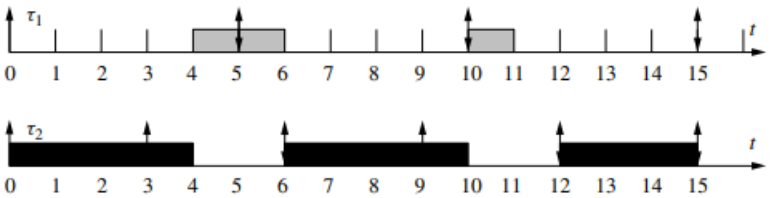
Základní myšlenkou, kterou tento model navrhuje, je, že chápeme periody a doby provedení jako pružné, nikoli jako fixní. Tyto dva parametry se chovají jako pružina, aby poskytovaly různou úroveň služeb. Úlohy mohou automaticky přizpůsobit svou periodu, aby systém správně pracoval při přetížení nebo pod zatížením. Je extrémně užitečný pro zpracování aplikací jako jsou multimédia, ve kterých musí být frekvence provádění některých výpočetních činností dynamicky upravena v závislosti na aktuálním stavu systému.

Každá úloha je charakterizována pěti parametry: časem výpočtu C_i, nominální periodou T_{i_0}, minimální periodou $T_{i_{min}}$, maximální periodou $T_{i_{max}}$ a elastickým koeficientem $e_i \geq 0$. Čím větší je $e_i \geq 0$, tím je úloha elastičtější. Elastickou úlohu tedy označujeme $\tau_i(C_i, T_{i_0}, T_{i_{min}}, T_{i_{max}}, e_i)$. Informace o tomto modelu byly čerpány z [3].

Multiframe model

Klasické přístupy předpokládají nejhorší dobu vykonávání úlohy (dále už jen WCET), což může být příliš konzervativní pro situace, kde průměrná doba provádění úloh je mnohem nižší než WCET. V multiframe modelu zachycujeme tuto variabilitu tím, že nespecifikujeme nejhorší možnou dobu provádění úlohy jako jedno číslo, ale jako konečný seznam čísel, ze kterých budou generovány doby provádění jednotlivých instancí úlohy. Konkrétně to tedy znamená, že opakováním tohoto seznamu čísel do nekonečna vzniká periodická posloupnost čísel, která zajišťuje, že čas provádění každé instance úlohy je omezen shora odpovídajícím číslem v periodické posloupnosti.

Úloha je charakterizována dvěma parametry: Γ je konečnou množinou N dob provádění $C^0, C^1, \ldots, C^{N-1}$ a minimálním časovým rozdílem P, který určuje, že časy, kdy je úloha ve stavu ready dvou po sobě jdoucích úloh, musí být minimálně N jednotek od sebe. Termín každého frame je P jednotek po tom, co je úloha připravená. Úlohu v multiframe modelu tedy značíme jako dvojici: $\tau_i(\Gamma, P)$. Informace o tomto modelu pochází z [12].



Obrázek 2.11: **Posloupnost provádění aplikace integrující dvě úlohy: jedna klasická úloha $\tau_1(0, 1, 5, 5)$ a jedna multiframe úloha $\tau_2(0, (3, 1), 3, 3)$** [12]

Statistical rate monotonic scheduling

Statistical rate monotonic scheduling (dále už jen jako SRMS) je obecnější verzí klasických výsledků rate monotonic scheduling. Tento přístup pracuje s periodickými úlohami s vysokou variabilitou dob provedení. Pro každou úlohu je definována úroveň služeb jako pravděpodobnost, že v libovolně dlouhé historii provádění bude náhodně vybraná instance této úlohy splňovat svůj termín. SRMS se skládá ze dvou částí: kontroléru přijetí úlohy

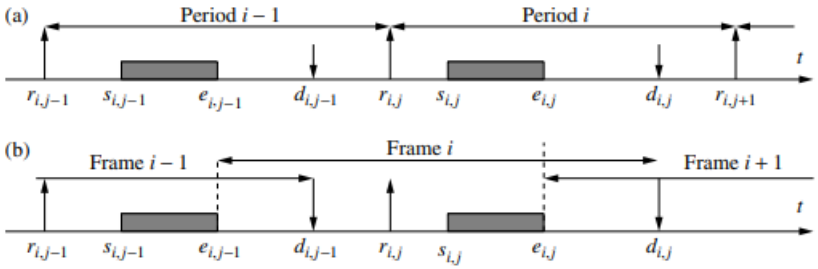
a plánovače. Kontrolér přijetí úlohy řídí úroveň poskytované služby různým úlohám prostřednictvím rozhodnutí o přijetí nebo odmítnutí a přiřazení priorit. Neplýtvá žádnými zdroji na instance úloh, které nestihnou své termíny kvůli přetížení. Plánovač je jednoduchý, preemptivní a má fixní priority. Použití tohoto modelu je vhodné například pro multimediální aplikace. Informace o tomto modelu byly čerpány z [1].

V SRMS je každé úloze přiřazen parametr a_i, který představuje dobu, po kterou je zdroj přiřazen této úloze během její superperiody. Proto podle SRMS platí nutná a postačující podmínka pro to, aby harmonická sada úloh byla plánovatelná:

$$\sum_{i=1}^{n} \frac{a_i}{P_{i+1}} \leq 1$$

Adaptive model

Adaptive model předpokládá, že termín úlohy je stanoven o jedno časové období po dokončení předchozí instance úlohy a čas uvolnění úlohy (anglicky release time) může být nastaven kdykoliv před termínem. Časová doména musí být rozdělena na rámce (anglicky frame) stejné délky. Hlavním cílem tohoto modelu je dosažení konstantního časového rozestupu mezi sousedními instancemi úlohy. Tento model podstatným způsobem snižuje variabilitu doby provádění, zatímco u klasických periodických úloh může tato variabilita dosáhnout nuly až dvojnásobku periody. Obrázek 2.12 ukazuje srovnání mezi klasickým modelem úlohy a adaptivním modelem úlohy. Základní rozdíl mezi oběma modely spočívá ve volbě časů uvolnění úlohy, které mohou být nastaveny kdykoliv před termínem v závislosti na individuálních požadavcích úlohy. Termín je tedy definován jako jedno časové období od dokončení předchozí instance úlohy. [5]



Obrázek 2.12: **Ilustrace Adaptive model** [5]

2.5 Dostupné nástroje

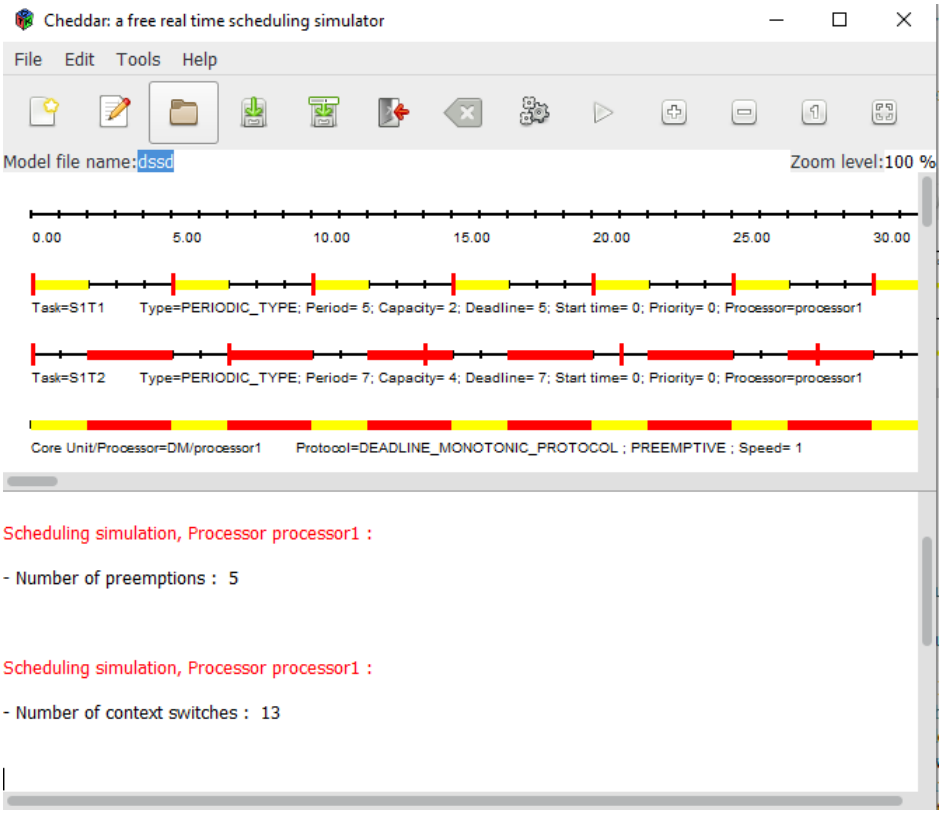
V dnešní době neexistuje mnoho alternativ nástrojů vhodných pro analýzu plánovatelnosti. V minulosti bylo možné vybírat z více variant a bylo potřeba zohlednit, co chceme sledovat, jelikož tyto nástroje byly často nesrovnatelné, a to především kvůli způsobu, jakým vyjadřovaly časové aspekty RT systémů [16]. Rozdíly mezi jednotlivými nástroji a jejich porovnání bylo zpracováno například v [13]. Během let byl vývoj většiny těchto nástrojů ukončen, a tak se dnes můžeme bavit hlavně o nástrojích UPPAAL, Cheddar a případně i Times, které si také popíšeme.

2.5.1 Cheddar

Cheddar je simulační a plánovací nástroj v reálném čase s licencí GNU GPL. Tento framework umožňuje modelování RT systémů a ověřování jejich plánovatelnosti nebo jiných výkonnostních kritérií.

Historie Cheddaru sahá až do roku 2002, kdy byl projekt Cheddar zahájen Frankem Singhoffem na Univerzitě v Brestu. Od roku 2008 se na vývoji podílí i společnost Ellidiss Technologies a poskytuje jeho průmyslovou podporu svým komerčním nástrojem nazvaným AADLInspector.

Skládá ze dvou nezávislých částí: editoru pro modelování systému a frameworku pro analýzu. Editor umožňuje popis systémů s jádry, procesory, jednotkami cache, síťovými čipy a softwarovými komponentami.



Obrázek 2.13: **Obrázek znázorňující nástroj Cheddar**

Cheddar umožňuje modelování systému s použitím AADL nebo vlastního interního jazyka CheddarADL. Experimenty probíhají také s použitím MARTE UML a dalších jazyků ADL, jako jsou PPOOA nebo MOSART. Cheddar hostí doménově specifický jazyk ve svém simulačním nástroji pro návrh nových modelů úloh nebo plánovacích politik.

Nástroj nám umožňuje větší flexibilitu v modelování nejistot než nástroj Times pomocí parametrů jednotlivých úloh. Je možné specifikovat parametry jako jitter, offset, initial offset, criticality a blocking time. Kdybychom chtěli implementovat jako zdroj nejistoty například přerušovací mechanismus, bylo by to příliš složité, a proto nevhodné pro naši implementaci Cheddar použít. V Cheddar také nelze využít prostředků SMC. Byl použit pouze na ověření výsledků sad, které v něm bylo možné namodelovat. Informace o nástroji byly převzaty z oficiálních stránek nástroje [17].

2.5.2 TimesTool

Times je nástroj pro modelování a analýzu plánovatelnosti pro vestavěné systémy reálného času. Nástroj byl vyvinutý na univerzitě Uppsala. Je vhodný pro systémy, které lze popsat jako soubor preemptivních nebo nepreemptivních úloh, které jsou periodicky nebo sporadicky spouštěny v určitém čase nebo externími událostmi. Poskytuje grafické rozhraní pro editaci a simulaci a engine pro analýzu plánovatelnosti.
Times se skládá ze čtyř hlavních částí:

- grafický editor pro časované automaty
- simulátor
- verifikátor pro analýzu plánovatelnosti
- generátor kódu pro automatickou syntézu C-kódu na platformě brickOS

Analýza plánovatelnosti v nástroji Times je založena na analýze dosažitelnosti automatu reprezentujícím plánovač, který je vytvořen podle vybraného plánovacího mechanismu. Automat je považován za plánovatelný v případě, že existuje (preemptivní nebo nepreemptivní) plánovací strategie, která umožňuje plánovat všechny možné sekvence událostí, jež jsou akceptovány tímto automatem. To znamená, že všechny související úlohy lze vykonat do jejich daných termínů.

Times neumožňuje modelování nejistot ani možnost specifikovat vetší množství parametrů úloh. Je možné pouze zvolit typ úlohy. Informace o nástroji byly čerpány z [21].

Kapitola 3

Realizační prostředky a návrh řešení

3.1 Zvolené realizační prostředky a metody

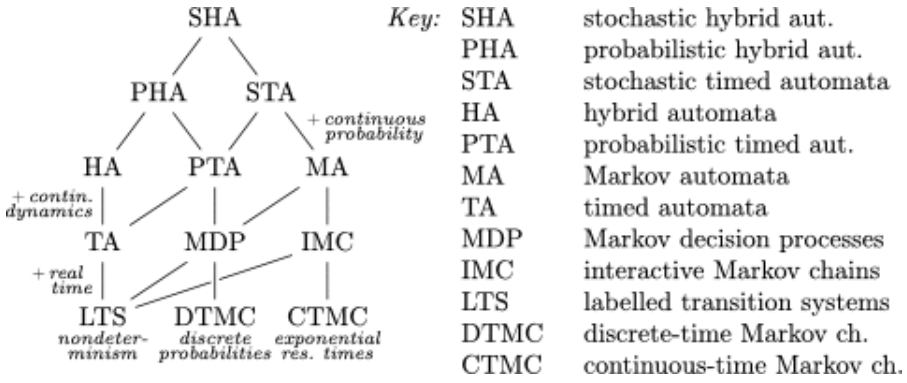
3.1.1 Statistical Model Checking

Statistical Model Checking (dále jen SMC) je přístup, který se používá jako alternativa k prohledávání stavového prostoru modelu. Zaměřuje se na sledování určitých simulací modelu a následně používá statistické výsledky, aby se mohlo rozhodnout, zda systém splňuje dotazovanou vlastnost s určitou jistotou. SMC je kompromisem mezi ověřováním modelu (anglicky Model Checking) a testováním. Metody založené na simulaci jsou méně náročné na čas a paměť a jsou často jedinou použitelnou variantou. SMC je implementováno v řadě nástrojů, které se používají na celou škálu případových studií. Tento přístup je využívaný v mnoha výzkumných oblastech, jako je automobilový průmysl, letecký průmysl, biologické systémy a také v softwarovém inženýrství, konkrétně pro průmyslové aplikace. Jedním z důvodů, proč slaví SMC takový úspěch, je jeho nenáročnost na implementaci, pochopení a použití. SMC umožňuje ověřování vlastností, které nelze vyjádřit v klasických temporálních logikách. [22]

3.1.2 Časované automaty

Časovaný automat je konečný stavový automat rozšířený o hodinové proměnné. Používá model, kde se hodinové proměnné vyhodnocují na reálné číslo a všechny hodiny postupují synchronně. Informace byly převzaty z [6]. *Časovaný automat* je definován jako uspořádaná šestice tvaru (L, l_0, C, A, E, I) kde:

- L je konečná množina stavů
- $l_0 \in L$ je počáteční stav
- C je konečná množina hodin
- A je konečná množina akcí
- E je konečná množina hran mezi stavy
- $E \subseteq L \times A \times B(C) \times 2^C \times L$ je množina hran mezi stavy s akcí



Obrázek 3.1: **Rodokmen kvantitativních automatů** [7]

3.1.3 Modální logika

Modální logika byla použita pro ověřování vlastností systému v nástroji UPPAAL (viz 3.1.4). Pomocí modální logiky dochází k rozšiřování výrokové a predikátové logiky. Na rozdíl od klasických logik nepracujeme jen s jediným světem, ale s množinou takzvaných konečných světů, kterou značíme W. Binární relace dostupnosti $R(w, u)$, často také zapisována jako wRu, říká, že svět u je dostupný ze světa w. R je množina uspořádaných dvojic světů z množiny W, tedy $R \subseteq W \times W$. Informace použité k modální logice byly převzaty z [14].

Kripkeho sémantika

Sémantika modální logiky se nazývá Kripkeho sémantika možných světů a jejím autorem je americký filozof a logik Saul Aaron Kripke. V sémantice modální logiky se pracuje s novými spojkami, které rozšiřují tradiční spojky používané v predikátové a výrokové logice. Nově ale zavádí spojky $\Box$ a $\diamond$, které se používají jako unární operátory. $\Box P$ znamená, že formule P platí ve všech možných světech. Naopak $\diamond P$ znamená, že formule P je pravdivá v alespoň jednom světě. Můžeme vidět souvislost s operátory $\forall$ a $\exists$ v predikátové logice a také říci, že obě tyto dvojice jsou komplementární. Z toto vyplývá, že $\Box P \equiv \neg \diamond \neg P$a $\diamond P \equiv \neg \Box \neg P$.

3.1.4 UPPAAL

UPPAAL je nástroj využívaný pro modelování, simulaci a verifikaci RT systémů, které jsou reprezentovány sítí časovaných automatů rozšířenou o proměnné, strukturované datové typy a synchronizaci kanálů. Skládá se ze tří částí: editoru, simulátoru a model-checkeru. Na jeho vývoji se podílí společně výzkumníci z švédské Uppsala University a dánské Aalborg University. UPPAAL byl uveden již v roce 1997 a od té doby se neustále pracuje na různých rozšířeních a verzích. Aktuálně je nejnovější dostupnou verzí verze 5.0. Informace použité v této sekci byly převzaty z [6] [22] [2].

Uppaal SMC

Pro verifikaci systémů reálného času nejsou časované automaty univerzálním řešením. Není možné zachytit chování složitých kyberneticko-fyzikálních systémů. Spojité chování těchto systémů často závisí jak na komplexní dynamice, tak na jejich stochastickém chování. Aproximace těchto chování pomocí časovaných automatů byla původně jediná možnost, která šla udělat.

UPPAAL SMC navrhuje alternativu k výše zmíněnému problému. Tato verze UPPAALu navrhuje reprezentovat systémy prostřednictvím sítí automatů, jejichž chování může záviset jak na stochastických, tak na nelineárních dynamických vlastnostech. Každá část systému je popsána automatem, jehož hodiny mohou běžet různou rychlostí. Tyto rychlosti lze specifikovat pomocí obyčejných diferenciálních rovnic. SMC je blíže popsáno v 3.1.1 a dotazy, které poskytuje verze SMC v 3.1.4.

Editor

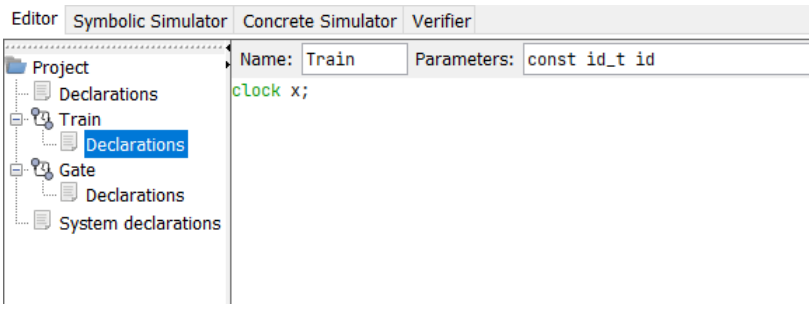
Jak už jsme v úvodu zmínili, v UPPAAL modelujeme systémy jako síť paralelních časovaných automatů rozšířených o hodinové proměnné, kdy všechny hodiny postupují synchronně. Model je navíc rozšířený o ohraničené diskrétní proměnné, které jsou součástí stavu. Využívají se stejně jako v ostatních programovacích jazycích, lze nad nimi provádět operace read a write a lze je používat jako členy v aritmetických operacích. Tyto proměnné mohou mít lokální nebo globální rozsah. Syntax a sémantika, která se používá pro deklaraci proměnných, vytváření funkcí a nebo například logických operací je podobná jazyku C.

```
bool foo(int a, int b)
{
    return a > b;
}
```

Obrázek 3.2: **Funkce v nástroji UPPAAL porovnávající dvě čísla**

Systém lze popsat pomocí:

- globální deklarace
- šablony a lokální deklarace
- definice systému



Obrázek 3.3: **Systémový editor nástroje UPPAAL**

Šablony, které reprezentují jednotlivé automaty mohou být definovány pomocí parametrů, které mohou být libovolného typu (bool, int, ...). Tyto parametry poté specifikujeme při deklaraci procesu - vkládáme argumenty jednotlivým šablonám v definici systému. Součástí šablon jsou také lokální proměnné, hodiny a funkce.

V UPPAALU rozlišujeme tyto stavy:

- Běžný
- Počáteční (anglicky Initial) – stav je počátečním stavem daného procesu. Počáteční stav může být zároveň commited nebo urgentní. Může také obsahovat invariant.
- Urgentní (anglicky Urgent) – tento stav je specifický tím, že v něm neběží čas. To znamená, že je to ekvivalentní situaci, kdy běžný stav rozšíříme o hodiny clock x. Všechny vstupní hrany by resetovaly tyto hodiny a stav by měl invariant $x \leq 0$.
- Commited – je přísnější než urgentní stav. Pokud je systém v alespoň jednom commited stavu, tak musí provést přechod přes hranu vedoucí z commited stavu.

(a) Běžný (b) Počáteční

(c) Urgentní (d) Commited

Obrázek 3.4: **Typy stavů v UPPAAL**

Procesy mezi sebou mohou komunikovat pomocí globálních proměnných nebo pomocí kanálů. Kanály rozdělujeme na:

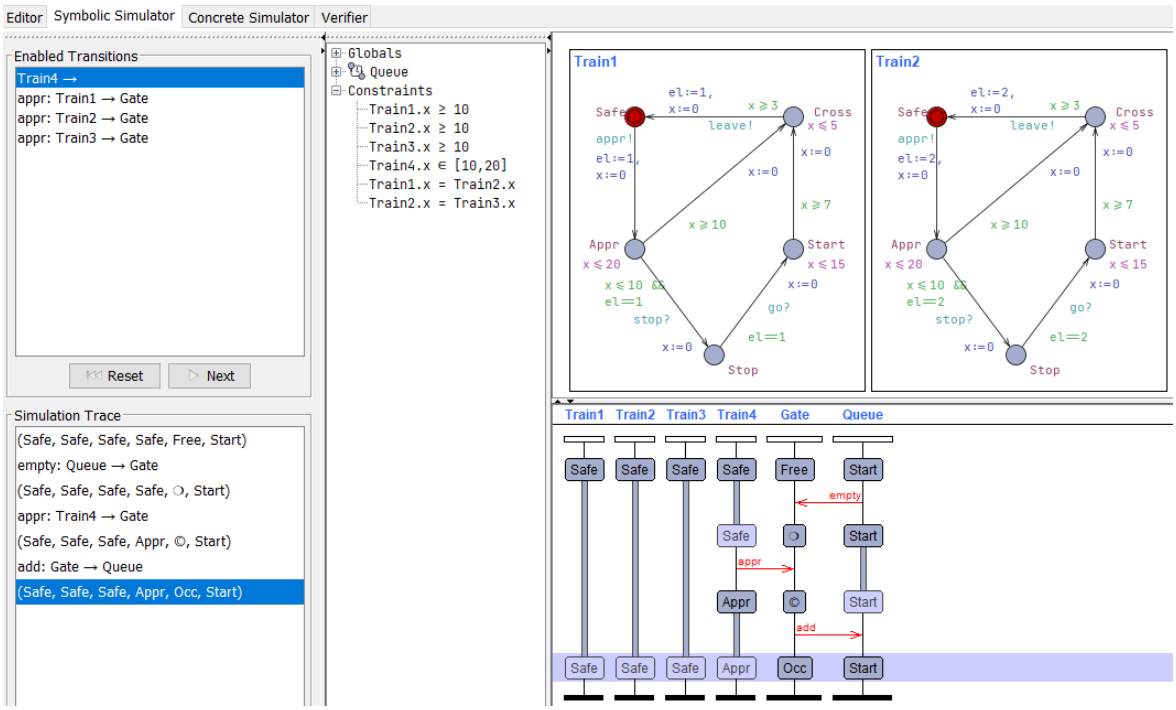
- Komunikační kanál – deklaruje se jako `chan c`. Hrana s označením `c!` se synchronizuje s hranou označenou `c?`. Pokud je možné vybrat několik takovýchto párů, výběr se provede nedeterministicky.
- Naléhavý kanál – při deklaraci se přidává před chan klíčové slovo `urgent`. Při synchronizaci přes tento druh kanálu nesmí nastat žádné zpoždění. Hrany, které používají naléhavé kanál, nemohou mít žádné časové omezení.
- Broadcast kanál – jsou deklarovány jako `broadcast chan c`. Při synchronizaci vysílání lze jednoho odesílatele `c!` synchronizovat s libovolným počtem přijímačů `c?`. Každý přijímač, který se může v aktuálním stavu synchronizovat, se synchronizovat musí. Pokud neexistují žádní příjemci, kteří by se synchronizovat mohli, může odesílatel stále provést akci `c!`. Z toho vyplývá, že broadcast vysílání není nikdy blokující.

Symbolic simulátor

Simulátor se používá jako validační nástroj, který umožňuje dynamické provádění systému během raných fází jeho návrhu nebo modelování. Tímto způsobem lze levně a jednoduše odhalit chyby před verifikací.

Simulátor lze použít třemi různými způsoby:

- Uživatel může spustit a ovládat systém a vybírat jednotlivé přechody systému
- Pomocí tlačítka `Random` lze přepnout do režimu, kdy systém běží samostatně. Lze nastavit rychlost jakou systém poběží a případně pozastavit běh a znovu přejít do manuálního režimu.
- Lze si krokovat pomocí záznamu běhu, který byl uložený nebo importovaný z verifikátoru, aby bylo možné ilustrovat dosažitelnost jednotlivých stavů.



Obrázek 3.5: **Symbolic simulátor**

Simulátor je rozdělen do čtyř částí (viz 3.5):

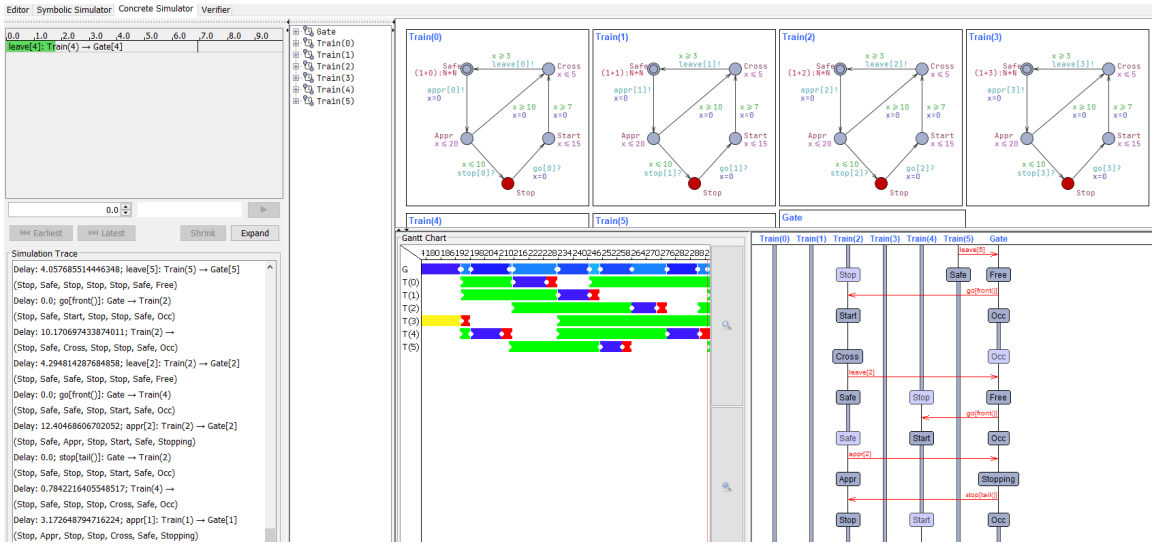
- Ovládací část – používá se k výběru a spouštění povolených přechodů, procházení běhů a přepínání náhodné (automatické) simulace.
- Zobrazení systému – zobrazuje všechny instance automatů a aktivní stavy v aktuálním stavu systému.
- Přehled proměnných – zobrazuje hodnoty jednotlivých proměnných a hodin, jak globálních, tak lokálních.
- Graf sekvencí zpráv – zobrazuje synchronizaci mezi jednotlivými procesy a aktivní stavy v jednotlivých krocích.

Concrete simulátor

Concrete simulátor je podobný symbolické simulaci tím, že jde o validační nástroj, který umožňuje zkoumat možné dynamické provádění systému. Rozdíl je v tom, že simulace je založena na konkrétních bězích. To znamená, že lze zvolit konkrétní čas pro provedení přechodu. Nástroj pomáhá zjistit, v jakém čase lze přechod provést. Obsahuje stejné části jako symbolic simulátor, a navíc obsahuje panel Ganttova Diagramu, který zobrazuje jiný pohled na generovaný běh podle specifikace Ganttova diagramu uvedené v definici systému.

Vodorovná osa v Ganttově diagramu představuje časový rozsah a ve svislé ose jsou uvedeny činnosti, obvykle některé ze systémových procesů, které jsou definovány v Ganttově diagramu. Svislá čára slouží k zobrazení aktuálního času. Vodorovné pruhy různých délek a barev reprezentují, kdy jsou různé výrazy ve specifikaci Ganttova diagramu splněny podle aktuálního stavu běhu.

Při kliknutí na dostupný přechod v Ganttově diagramu se zobrazí informace o intervalech, kdy jsou splněny příslušné výrazy.



Obrázek 3.6: **Concrete simulator**

Verifikátor

Verifikátor slouží ke ověřování vlastností systému. Dotazy na systém musí být vyjádřeny formálně dobře definovaným a strojově čitelným jazykem. Uppaal používá zjednodušenou verzi Computation Tree Logic (dále už jen CTL). Podobně jako v CTL se jazyk skládá ze stavových formulí a formulí, které jsou definované v rámci přechodů mezi stavy, které tvoří cesty. Stavová formule je výraz, který lze vyhodnotit pro stav bez ohledu na chování modelu. Například to může být jednoduchý výraz, jako je i == 7, který je pravdivý ve stavu, když i je rovno 7.

V standardní verzi UPPAAL lze ověřovat tři druhy vlastností:

- Živost – tyto vlastnosti lze popsat jako: něco se nakonec stane. Například při stisknutí tlačítka zapnutí na dálkovém ovladači televize by se nakonec televize měla zapnout. Jednoduše lze živost vyjádřit jako formuli $A <> \varphi$, což znamená, že φ je nakonec splněno. Vhodnějším způsobem, jak vyjádřit živost, je pomocí formule $\varphi -> \psi$. To bychom mohli popsat jako: když je splněno φ, pak bude splněno i ψ.
- Dosažitelnost – tyto vlastnosti se ptají, zda je možné dosáhnout stavu, který splňuje danou podmínku, a jsou často používány pro ověření základního chování modelu. Jinak řečeno: ptají se, zda lze stavovou formuli φ splnit v jakémkoli dosažitelném stavu.V UPPAAL dotaz na tuto vlastnost zapisujeme pomocí syntaxe $E <> \varphi$.
- Bezpečnost – bezpečnostní vlastnosti slouží k ověření, že nějaká věc se nikdy nestane. V UPPAAL tyto vlastnosti formulujeme pozitivně, což znamená, že nějaká věc je invariantně pravdivá. Nechť φ je stavová formule. Vyjadřujeme, že φ by mělo být pravdivé ve všech dosažitelných stavech pomocí formule $A[\,]\ \varphi$, zatímco $E[\,]\ \varphi$ říká, že by měla existovat maximální cesta, na které je φ vždy pravdivé.

Kromě standardních dotazů poskytuje UPPAAL SMC několik nových dotazů souvisejících se stochastickou interpretací časovaných automatů. Umožňuje uživateli vizualizovat hodnoty výrazů, vyhodnocených na celá čísla nebo hodiny během simulovaných běhů.
To poskytuje uživateli náhled na chování systému, a tak se může dotazovat i na zajímavější vlastnosti. Syntax v UPPAAL SMC vypadá následovně:

```
simulate N [<=bound] { E1, .., Ek }
```

`N` je přirozené číslo, které udává počet simulací, které budou provedeny. `bound` je časová mez jednotlivých situací a `E1, .., Ek` jsou stavové výrazy, které budou sledovány a vizualizovány.

Na následujících příkladech si ukážeme a vysvětlíme různé konstrukce dotazů. Dotazy se vztahují k systému popsaném v 3.1.4.

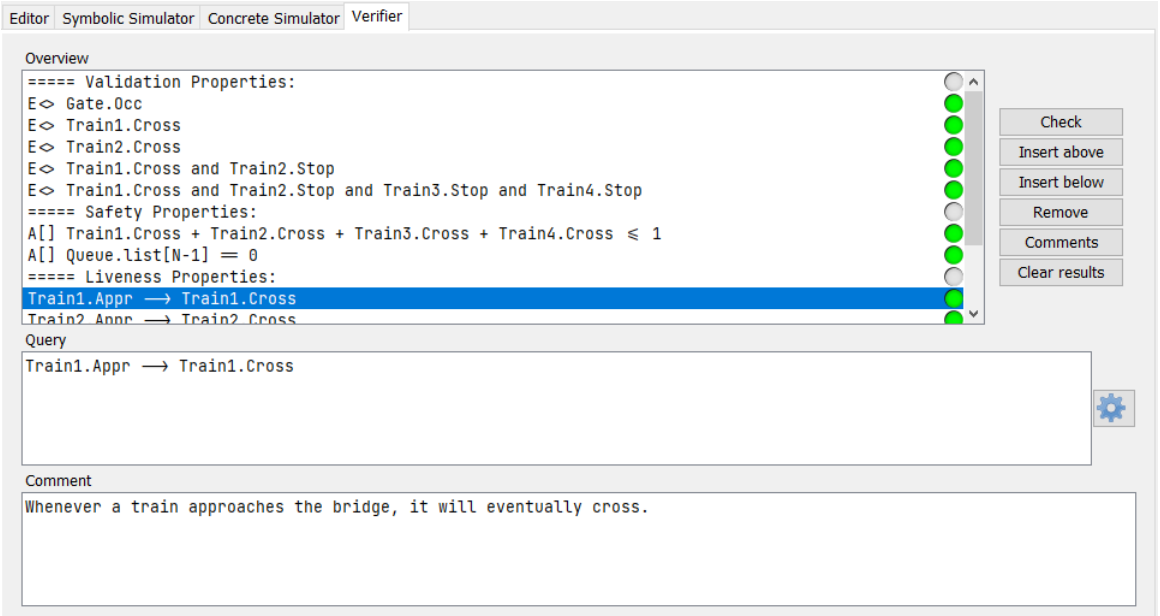
- `A[] Train(0).Cross+Train(1).Cross`$\leq$`1` – Nikdy neprojíždí mostem více než jeden vlak. Tento výraz využívá skutečnost, že Train(1).Cross se vyhodnocuje jako true nebo false, tj. 1 nebo 0.
- `A[] not deadlock` – Tímto výrazem ověřujeme, že v systému nedojde k uváznutí.
- `Pr[<=100] (<> Train(0).Cross) >= 0.2` – Ověřuje, zda pravděpodobnost, že vlak 0 přejede most během 100 časových jednotek, přesahuje 0,2.
- `E[<=20; 20000](max: sum(i:id_t) Train(i).Stop)` – S využitím 20 000 běhů chceme znát průměr maximálního počtu vlaků, které jsou zastaveny během prvních 20 časových jednotek.
- `simulate 1 [<=300] {Train(0).Cross, Train(5).Cross, Gate.len}` – Pomocí tohoto výrazu můžeme sledovat, kdy vlak 0 a vlak 5 přejíždějí most, stejně jako délku fronty.

Specifikace požadavků

Jednotlivé požadavky (anglicky queries) jsou zobrazeny na panelu `Overview` ve verifikátoru. Pomocí tlačítka `Comments/Queries` lze přepínat, jestli se v panelu `Overview` budou zobrazovat požadavky nebo komentáře k nim.

Dotazy můžeme vybírat kliknutím levého tlačítka myši jeden po druhém, nebo pokud chceme požadavky vybírat najednou, je potřeba využít kombinaci myši a kláves Shift a Ctrl. Pomocí klávesy Shift zvolíme rozsah a pomocí klávesy Ctrl tento rozsah zrušíme. První vybraný požadavek se zobrazí v polích `Query` a `Comment`, kde je možné požadavek nebo komentář k němu i upravit.

Pro přidávání a odebírání nových záznamů slouží tlačítka `Insert` a `Delete`. `Delete` slouží k odstranění všech označených záznamů v poli `Overview`. Pokud není označený žádný záznam, nic se nevymaže.



Obrázek 3.7: **Verifikátor**

Verifikace požadavků

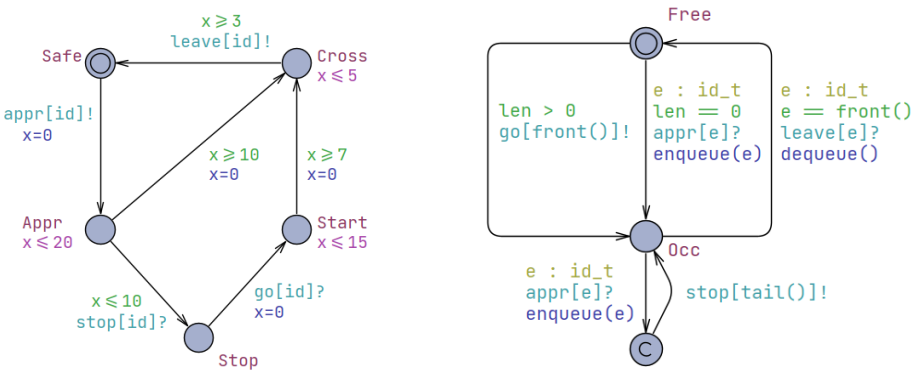
Verifikace je provedena podle nastavení specifikovaného v `Options menu`. Označené požadavky jsou verifikovány při kliknutí na tlačítko `Check`. Poté se zobrazí dialogové okno zobrazující informace o probíhající verifikaci jako například, kolik požadavků již bylo verifikováno nebo aktuální využitý procesorový čas. Jednotlivé údaje zobrazené v tomto okně se mohou lišit dle operačního systému.

Výstup verifikátoru je zobrazen v poli `Status`. Výsledek je také označen kruhovými značkami ve sloupci zcela vpravo na panelu `Overview`. Šedá barva znázorňuje, že pravdivostní hodnota vlastnosti je neznámá, zelená barva, že vlastnost je splněna, a červená barva, že vlastnost splněna není.

Příklad systému

Systém s vlakem a závorou je součástí UPPAALu jako jeden z příkladů pro pochopení jeho fungování. Jedná se o řídicí systém železnice, který řídí vjezd na most pro několik vlaků. Most je kritický sdílený zdroj, ke kterému může mít přístup pouze jeden vlak. Systém je definován jako několik vlaků a kontrolér.

Šablona vlaku Šablona na obrázku 3.8 má pět stavů: `Safe, Appr, Stop, Start a Cross`. Počátečním stavem je `Safe`, kde se vlak ještě nepřibližuje a může zde zůstat libovolně dlouho. Při příjezdu vlaku se synchronizuje s kontrolérem pomocí kanálu `appr[id]!` při přechodu do stavu `Appr` a hodiny `x` se resetují. Stav `Appr` má invariant `x`$\leq$`20`, což znamená, že tento stav být opuštěn do 20 časových jednotek. Vlak nemůže být okamžitě zastaven a rozjetí také nějakou dobu trvá. Přechody se stráží `x`$\leq$`10` a `x`$\geq$`10` odpovídají situacím, kdy může být vlak zastaven, nebo už zastaven být nemůže. Pokud vlak zastaven není což může nastat v čase mezi 10 a 20 časovými jednotkami, přechází do stavu `Cross`, který bude popsán později. Vlak může být zastaven pouze do 10 časových jednotek, pokud dostane signál od kontroléru, přechází do stavu `Stop`. V stavu `Stop` se čeká na synchronizaci `go[id]?`, aby se vlak znovu rozjel. Dostáváme se do stavu `Start` a resetujeme hodiny `x`. Stav `Start` dovolí vlaku se rozjet mezi 7 a 15 časovými jednotkami. Stav`Cross` je opuštěn mezi 3 a 5 časovými jednotkami po vstupu. Poté, co vlak opustil most, posílá kontroléru signál `leave[id]!`.



(a) Vlak (b) Brána

Obrázek 3.8: **Šablona vlaku a brány**

Šablona brány Kontrolér brány na obrázku 3.8 se synchronizuje s vlaky. Kontrolér začíná v počátečním stavu `Free`, což představuje stav, kdy je most je volný. V tomto stavu zjišťuje stav fronty, zda je prázdná nebo ne. Pokud je fronta prázdná, pak kontrolér čeká na přibližující se vlaky se synchronizací `appr[e]?`. Když vlak přijíždí, je přidán do fronty pomocí funkce `void enqueue(id_t element)`. Pokud fronta není prázdná, pak je prvnímu vlaku ve frontě dán signál k rozjezdu synchronizací `go!`. V místě `Occ` kontrolér v čeká, až běžící vlak opustí most `leave[e]?` a pomocí funkce `void dequeue(id_t element)` jej odebírá z fronty. Pokud se k němu ale přibližují další vlaky `appr[e]?`, jsou přidány do fronty a zastaveny `stop!`. K tomuto je využíván nepojmenovaný commited stav.

3.2 Návrh řešení

Tato sekce popisuje návrh řešení a požadavky na řešení.

3.2.1 Požadavky

Cílem je vytvoření přístupu k plánovatelnosti úloh reálného času s ohledem na nejistotu. Při implementaci má být využit SMC a jeho prostředky (zmíněné v 3.1). Dále by měly být vytvořeny sady úloh a provedení analýzy plánovatelnosti těchto sad pomocí navrženého přístupu.

3.2.2 Návrh přístupu

Abstraktní model

Celý proces začíná vytvořením abstraktního modelu. Je potřeba určit, jaké budou RT úlohy obsahovat parametry, jestli bude plánovač preemptivní nebo nepreemptivní nebo jaké plánovací mechanismy chceme použít. Dále je potřeba brát v potaz, jestli máme jeden zdroj prostředků nebo více a jestli bude probíhat plánování ve více frontách, nebo budeme používat pouze jednu frontu. Jelikož při analýze zohledňujeme i nejistotu, je nutné určit, co v našem systému bude nejistotu představovat. Je tedy potřeba si specifikovat systém a vytvořit abstraktní model, který bude podkladem pro simulační model. Nejistotou by mohly být například systémová přerušení.

Vytvoření simulačního modelu

Dalším krokem je vytvoření simulačního modelu v nástroji UPPAAL SMC. Model musí pro správný výstup simulace přesně zachytit chování systému včetně jeho úloh, plánovačů, plánovacích mechanismů a nejistot. Model se bude skládat z několika mezi sebou komunikujících modelů. Každý model bude reprezentovat jednotlivou část systému. Model úlohy by měl v každém případě obsahovat stav reprezentující chybový stav, do kterého se úloha dostane, pokud dojde k překročení časových mezí.

Specifikace požadavků na systém

Dále je důležité v UPPAAL SMC specifikovat požadavky na systém, které chceme ověřovat. Základní sledovanou vlastností z pohledu analýzy plánovatelnosti je, jestli se některá z úloh nedostala do chybového stavu zmíněného u simulačního modelu. Dále lze například sledovat, jestli nedošlo k deadlocku a jestli se všechny úlohy dostaly do stavu znázorňujícího dokončení úlohy. Lze také zjišťovat, jaká je pravděpodobnost, že k překročení mezí u konkrétní úlohy dojde.

Analýza výsledků

Na základě analýzy výsledků zjišťujeme, jestli je sledovaná sada úloh plánovatelná a pomocí jakých mechanismů. Na základě výsledků lze také upravit jednotlivé parametry úloh pro zpřesnění výsledků nebo pro dosažení plánovatelnosti.

Kapitola 4

Popis řešení

Tato kapitola se věnuje nejdříve popisu datových struktur a typů, které jsou pak využívány jednotlivými modely použitými pro implementaci frameworku. Jako implementační nástroj byl zvolen nástroj UPPAAL (popsaný v sekci 3.1.4), konkrétně jeho verze SMC. Popíšeme si jednotlivé stavy, přechody a všechny důležité informace a specifika každého modelu. Na závěr této kapitoly si popíšeme jednotlivé vlastnosti, které jsou u sad úloh ověřovány. Implementace zmíněná v této sekci realizuje navržený přístup (popsaný v sekci 3.2.2).

4.1 Datové typy

V této sekci se budeme věnovat vlastním definovaným datovým typům, které byly vytvořeny pro větší přehlednost a práci s frameworkem.

> **task_id**: ID úlohy typu integer s hodnotou od 1 po `task_number`, což je globální konstanta reprezentující počet úloh.

V případě, že plánujeme více frontově, pracujeme ještě s dalším datovým typem:

> **queue_id**: ID fronty typu integer s hodnotou od 1 po `queue_number`, což je globální konstanta reprezentující počet front.

4.2 Datové struktury

V této sekci se budeme věnovat strukturovaným datovým typům, které byly vytvořeny pro jednodušší a přehlednější manipulaci a reprezentaci úlohy a fronty. Nejdříve si popíšeme strukturu `task_t` a poté strukturu `queue_t`.

4.2.1 task_t

Struktura `task_t` reprezentuje RT úlohu a její parametry. Je definována globálně a lokálně jsou implementovány funkce pro získávání parametrů úlohy. Jednotlivé úlohy ukládáme před ověřováním vlastností do globálního konstantního pole `const task_t task[taskcount]`.

Struktura obsahuje tyto prvky:

- `int task_priority` – statická priorita úlohy předem definována uživatelem, využívá se pouze u fixed priority scheduling jinak je její hodnota 0.
- `int initial_offset` – délka zpoždění při prvním příchodu úlohy.
- `int min_period` – minimální délka periody dané úlohy, po uplynutí této doby může dojít k vytvoření další instance úlohy. U aperiodické úlohy je hodnota nastavena na 0, protože se nepoužívá.
- `int max_period` – maximální délka periody úlohy do které musí dojít k vytvoření další instance úlohy, používá se pouze u periodických úloh, sporadické a aperiodické úlohy mají tento atribut nastaven na hodnotu 0.
- `int offset` – zpoždění úlohy, které nastane každou periodu před uvolněním úlohy.
- `int deadline` – časová mez úlohy, při překročení úloha přechází do chybového stavu.
- `int bcet` – nejlepší doba provádění úlohy, až po uplynutí této doby může dojít k dokončení úlohy.
- `int wcet` – nejhorší doba provádění úlohy, do této doby musí dojít k dokončení úlohy.
- `queue_id queue` – jednoznačný identifikátor fronty do které úloha patří.

```
typedef struct {
  task_id id
  int task_priority;
  int initial_offset;
  int min_period;
  int max_period;
  int offset;
  int deadline;
  int bcet;
  int wcet;
  buffer_id queue;
} task_t;
```

Obrázek 4.1: **Struktura task_t v UPPAAL**

Některé z parametrů představují nejistotu pro systémy RT. Například parametry `InitialOffset` a `Offset` nebo nedeterministická doba běhu úlohy dána intervalem od `BCET` do `WCET` a také nedeterministická délka periody dána intervalem od `min_period` do `max_period`.

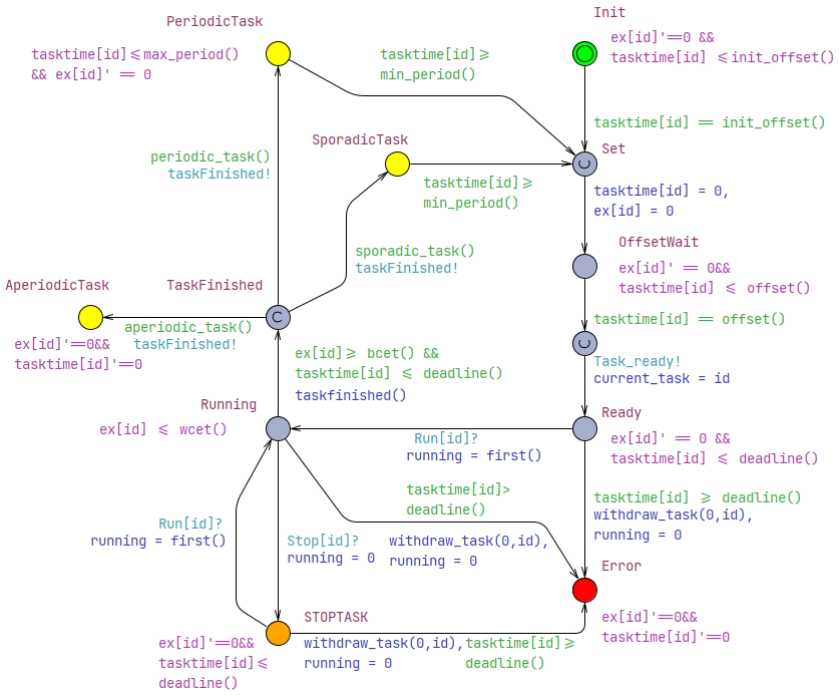
4.2.2 queue_t

Pro reprezentaci fronty byla vytvořena struktura `queue_t`. Struktura má pouze dva prvky: `int lenght` a pole `int nodes[task_number]`. Všechny fronty jsou uloženy v globálním poli nazvaném `queues`.

4.3 Modely v UPPAAL SMC

4.3.1 Model úlohy

Nejdříve si popíšeme model úlohy, který je zobrazen na obrázku 2.12 a používá se pro všechny typy úloh. Pracuje nad reprezentací úlohy, strukturou `task_t`. Lokálně jsou deklarovány pomocné funkce na získávání parametrů úlohy.



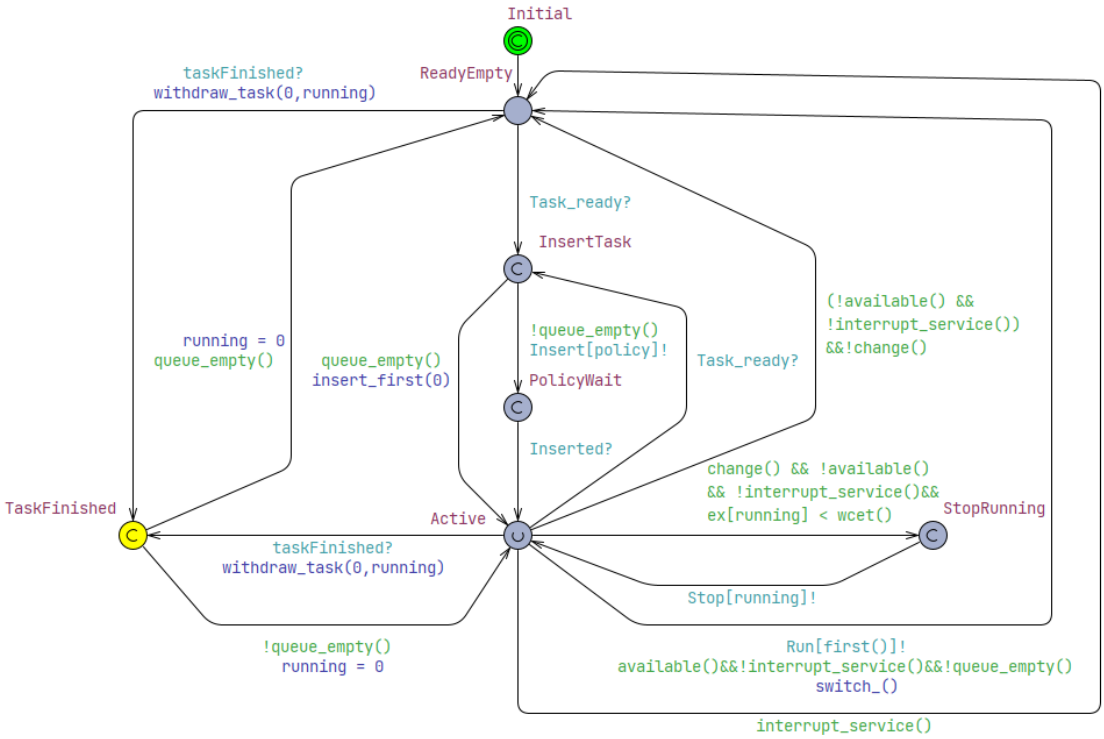
Obrázek 4.2: **Model úlohy**

Úloha začíná v počátečním stavu `Init`. Stráž a invariant nutí úlohu setrvat v tomto stavu, dokud hodiny nedosáhnou hodnoty dané parametrem `initial_offset`. V `Init` se nachází vždy jen první instance úlohy. Z `Init` úloha přechází do urgentního stavu `Set`, který v nulovém čase opouští, přechází do stavu `Offset` a nuluje globální hodiny `tasktime[id]` a `ex[id]`. Hodinová proměnná `ex[id]` reprezentuje dobu provádění úlohy a běží pouze ve stavu `Running`. K pozastavení běhu hodin je použit invariant `ex[id]'==0`. Stav `Offset` znovu nutí úlohu zůstat v tomto stavu, dokud nedojde k splnění stráže. Poté, co `tasktime[id]` dosáhne hodnoty parametru `Offset`, oznamuje úloha plánovači pomocí synchronizace `Ready!`, že je úloha připravená, nastavuje globální proměnnou `current_task` na svoje id a přechází do stavu `Ready`. Z `Ready` přechází do stavu `Running`, pokud obdrží signál `Run[id]?` od plánovače. V `Running` se může nacházet vždy jen jedna aktuálně běžící úloha. Pokud v `Running` obdrží signál `Stop[id]?` od plánovače, přechází do `STOPTASK`, který reprezentuje pozastavenou úlohu. Při překročení časové meze úloha přechází ze stavů `Ready`, `Running` nebo `STOPTASK` do stavu `Error`. Pokud je čas úlohy v mezích hodnot `BCET` a `WCET` a zároveň nedošlo k překročení časové meze, přechází úloha do stavu `Finished`.

Na základě typu úlohy poté přechází do stavů `PeriodicTask`, `SporadicTask` nebo `AperiodicTask`. Pokud se jedná o úlohu periodickou, další instance přichází v čase daném intervalem od `min_period` do `max_period`. U sporadické úlohy je dána pouze spodní hodnota intervalu hodnotou `min_period`. Aperiodická úloha má vždy jen jednu instanci.

4.3.2 Model plánovače

Šabloně zadáváme parametr `int policy`, který určuje, kterému plánovacímu mechanismu má plánovač signalizovat, aby vložil úlohu do fronty.

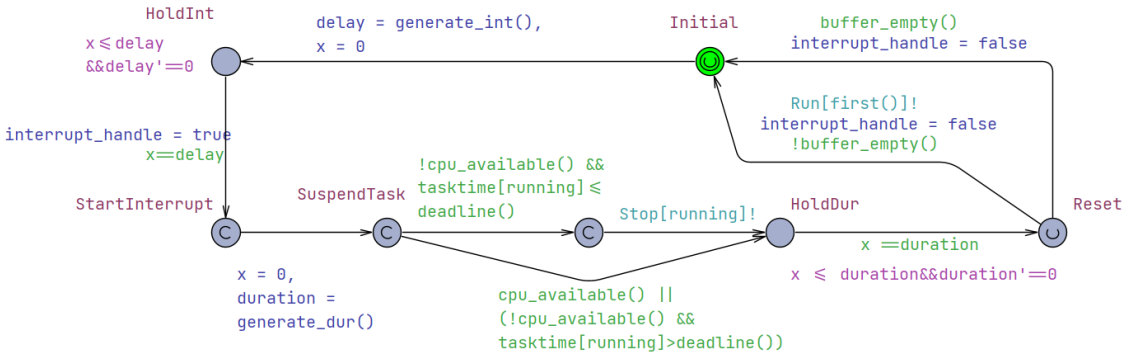


Obrázek 4.3: **Model plánovače**

Model začíná v počátečním comitted stavu `Initial`, z kterého přechází do stavu `ReadyEmpty`, kde čeká na synchronizaci s připravenou nebo dokončenou úlohou. Připravená úloha se synchronizuje pomocí kanálu `Task_ready?`. Pokud je fronta prázdná, což zjišťuje pomocí stráže `queue_empty()`, provede se přechod do urgent stavu `Active`. Pokud prázdná není, plánovač se synchronizuje s plánovacím mechanismem pomocí `Insert[policy]!` a přechází do comitted stavu `PolicyWait`. Zde čeká na potvrzení o vložení úlohy pomocí synchronizace `Inserted?` a přechází do `Active`. Pokud nedochází k obsluze přerušení nebo žádné úloze není přidělen procesor, pomocí synchronizace `Run[first()]!` dává signál k běhu běžící úloze. Pokud se v systému nachází běžící úloha a dojde ke změně úlohy s nejvyšší prioritou, běžící úlohu pozastaví synchronizací `Stop!`. Poté pomocí synchronizace `Run[first()]!` přiděluje procesor úloze s nejvyšší prioritou. V případě, že obdrží od úlohy synchronizaci pomocí `taskFinished[first()]?`, odebere dokončenou úlohu z fronty a dostane se do comitted stavu `TaskFinished`. Podle toho, jestli je fronta prázdná nebo ne, se provede přechod do `Active` a nebo do `Ready`.

4.3.3 Model přerušení

Model přerušení je založen na dvou *stop-watch* automatech. Ty umožňují generovat a provádět akce podle určitého časového harmonogramu. První automat generuje přerušení a druhý měří časový interval mezi spuštěním a ukončením přerušení.



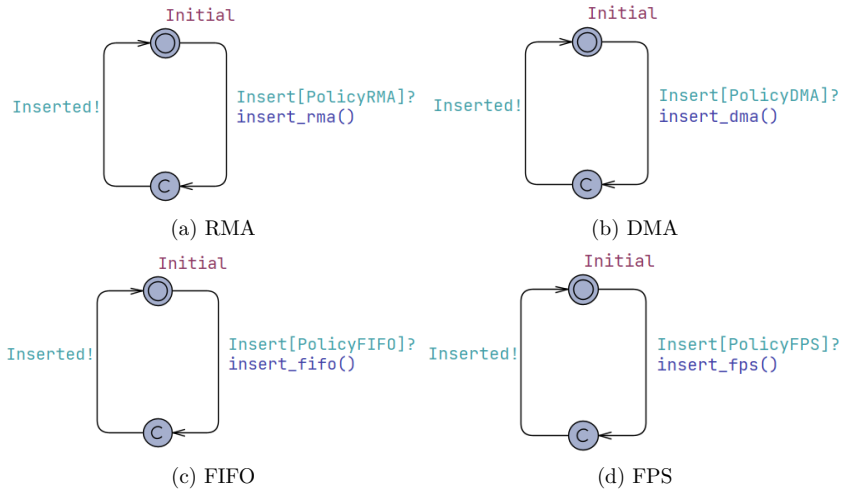
Obrázek 4.4: **Model přerušení**.

Počátečním stavem je commited stav `Initial`. Přechod do stavu `HoldInt` nuluje hodinovou proměnnou `x` a generuje dobu příchodu přerušení funkcí `generate_int()`. V `HoldInt` setrvává po dobu danou `delay`, což zajišťuje invariant `x <= delay && delay'==0` a stráž `x==delay`. Zároveň se při přechodu do commited stavu `StartInterrupt` nastavuje globální proměnná `bool interrupt_handle` na true. Tím zabraňujeme plánovači spouštět úlohy. Z `StartInterrupt` přechází do commited stavu `SuspendTask`, nuluje hodinovou proměnnou `x` a generuje dobu trvání přerušení pomocí funkce `generate_dur()`. V stavu `SuspendTask` se rozhoduje, který přechod bude využit podle toho, jestli se v systému nachází běžící úloha. Zjišťujeme to pomocí funkce `cpu_available()`. Pokud se v systému běžící úloha nachází, provede se přechod do nepojmenovaného commited stavu, proběhne synchronizace `Stop[running]!` s aktuálně běžící úlohou a přechod do `HoldDur`. V druhém případě se jen provede přechod do stavu `HoldDur`. V něm setrvává dokud `x` nedosáhne doby dané `duration`. Toto chování stejně jako v stavu `HoldInt` zajišťuje invariant a stráž. Z `HoldInt` přechází do commited stavu `Reset`. Při přechodu do počátečního stavu `Initial` v obou přechodech nastavuje `interrupt_handle` na true. Pokud je fronta prázdná, dává pokyn první úloze ve frontě k běhu pomocí synchronizace `Run[first()]!`.

4.3.4 Modely plánovacích mechanismů

RMA, DMA, FPS, FIFO

Modely pro mechanismy zobrazené na obrázku 4.5 si jsou podobné a fungují na stejném principu. Skládají se pouze ze dvou stavů: počáteční `Init` a commited stavu `TaskInserted`. Mechanismy jsou synchronizovány s plánovačem pomocí globálních kanálů `Insert[policy]?`. Po přijetí signálu, že se má využít daný mechanismus se provede přechod a odpovídajícím způsobem se vloží úloha do fronty.



(a) RMA (b) DMA

(c) FIFO (d) FPS

Obrázek 4.5: **Modely mechanismů RMA, DMA, FIFO, FPS**

Pro každý mechanismus je vytvořena funkce na vkládání úloh do fronty. Funkce odpovídají plánovacím mechanismům vysvětlených v 2.3. Funkce `insert_fifo()` pouze vkládá úlohu na poslední místo ve frontě. Zbylé tři funkce fungují na jiném principu. Funkce prohledávají frontu od začátku, pokud nastane situace, že se na pozici nachází úloha s menší prioritou, na danou pozici vloží úlohu a zbytek fronty posune o jednu pozici dál.

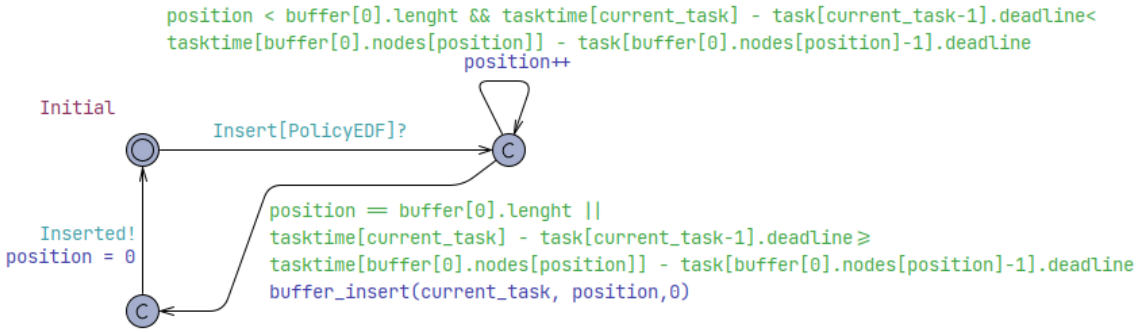
Po provedení přechodu a vložení úlohy na správnou pozici se úloha vrací do stavu `Init` a pomocí kanálu `Inserted!` oznamuje plánovači, že úloha byla úspěšně vložena.

```
void insert_rma()
{
    int bid = get_buff(current_task);
    int i;
    for( i=(preemptive ? 1 : 0); i < buffer[bid].lenght; i++)
    {
        if(task[current_task-1].maxperiod < task[buffer[bid].nodes[i]-1].maxperiod)
        {
            buffer_insert(current_task, i,bid);
            return;
        }
    }
    buffer_insert(current_task, buffer[bid].lenght,bid);
}
```

Obrázek 4.6: **Funkce insert_rma()**

EDF

Implementace mechanismu EDF byla složitější v porovnání s ostatními mechanismy a to z toho důvodu, že je potřeba pracovat s hodinovou proměnnou. V nástroji UPPAAL nelze s datovým typem `clock` pracovat ve funkcích. Kvůli tomu není možné implementovat tyto modely pouze jako dva stavy s funkcí pro vložení na správnou pozici. Modely obsahují tři stavy: `Init, Iterate` a `TaskInserted`. Provádíme ten stejný proces, jen trochu jiným způsobem. Po signálu od plánovače pomocí kanálu `Insert[policy]?` přechází model do commited stavu `Iterate`. Pokud je na pozici dané proměnnou `position` úloha s větší prioritou, provedeme „self-transition" přechod, kde inkrementujeme proměnnou `position`. Pokud se již nacházíme na konci fronty nebo je na dané pozici úloha s nižší prioritou, provedeme přechod do commited stavu `TaskInserted` a vložíme úlohu na pozici danou proměnnou `position`. Z tohoto stavu se hned přesouváme zpět do stavu `Init` a pomocí kanálu `Inserted!` dáváme signál plánovači o tom, že byla úloha vložena do fronty.

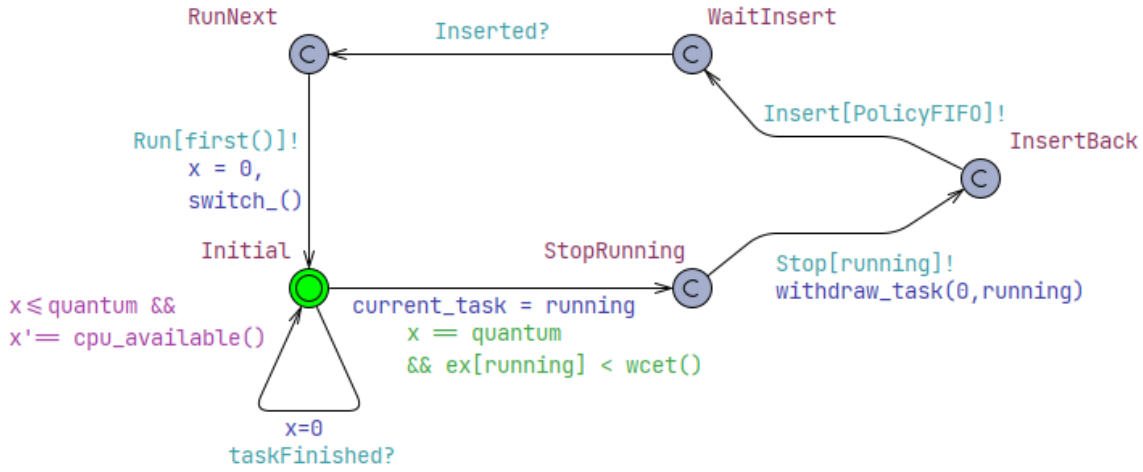


Obrázek 4.7: **EDF mechanismus**

Round robin

Jako poslední si z plánovacích mechanismů popíšeme Round robin. Šabloně zadáváme jeden parametr `int quantum`, kterým určujeme délku časového kvanta, které bude přiděleno jednotlivým úlohám. Aby tento mechanismus správně fungoval, musí se využívat spolu s modelem mechanismu FIFO pro vkládání úloh do fronty.

Počátečním stavem je `Initial` a k měření délky kvanta se používají lokální hodiny `clock x`. Invariant `x` $\leq$ `quantum && x'==isrunning()` zajišťuje setrvání v stavu po dobu trvání kvanta a běh hodin `x` jen v případě, že nějaká úloha běží. Přechod z počátečního stavu nastane jen ve dvou případech. V prvním případě se dokončí běžící úloha před vypršením kvanta, což sleduje pomocí synchronizace `taskFinished[running]?` a provede se „self-transition" přechod, v rámci kterého vynuluje hodiny `x`. Signál k běhu další úloze pak dává plánovač. V druhém případě dojde k vypršení časového kvanta. Model přejde do comitted stavu `StopRunning` a odebere se úloha z čela fronty. Pomocí synchronizace `Stop!` pozastavuje aktuálně běžící úlohu a přechází do stavu `InsertBack`. Zde proběhne synchronizace s plánovacím mechanismem FIFO pomocí `Insert[PolicyFIFO]!`, který vloží úlohu na konec fronty. V stavu `WaitInsert` čeká na synchronizaci s FIFO pomocí `Inserted?`. Synchronizace signalizuje, že byla úloha úspěšně vložena na konec fronty. Poté se dostane do commited stavu `RunNext`, ze kterého přechází zpět do počátečního stavu a provádí synchronizaci `Run[first()]!`, čímž dává signál k běhu první úloze ve frontě.



Obrázek 4.8: **Round robin mechanismus**

4.4 Vlastnosti k ověření

Na závěr této kapitoly představíme jednotlivé vlastnosti, které budeme ověřovat. Získané výsledky slouží jak pro kontrolu správnosti chování modelu, tak i pro analýzu plánovatelnosti úloh a také k získávání informací pro lepší porozumění chování systému v různých situacích.

4.4.1 Standardní dotazy

- `A[] forall (i : task_id) forall (j : task_id) Task(i).Running && Task(j).Running imply i == j` – výraz ověřuje, zda v žádném okamžiku neběží více než jedna úloha. Jedná se o důležitý validační dotaz, jelikož tato situace by nikdy nastat neměla.
- `A[] not deadlock` – tímto výrazem ověřujeme, že v systému nedojde k uváznutí.
- `A[] forall (i : task_id) not Task(i).Error` – výraz ověřuje, jestli se nějaká z úloh nedostane do stavu `Error`. Jedná se o hlavní otázku na systém. Platný predikát zaručuje definitivní plánovatelnost sady úkolů, zatímco negativní hodnocení predikátu naznačuje, že sada úkolů není plánovatelná.
- `A<> forall (i : task_id) Task(i).AperiodicTask` – u aperiodických sad tímto výrazem budeme ověřovat, zda se všechny úlohy dokončily a nachází se ve stavu `AperiodicTask`.

4.4.2 Statistické dotazy

U statistických dotazů byly některé parametry pro zajištění vyšší přesnosti nastaveny na jiné než výchozí hodnoty. Konkrétně se jedná o parametry: pravděpodobnost falešně pozitivních ($\alpha = 0.01$), pravděpodobnost falešně negativních ($\beta = 0.01$) a pravděpodobnostní nejistota ($\varepsilon = 0.1$).

- `E[<=X; Y](max: ex[i])` – s využitím Y běhů chceme znát maximální dobu provádění úlohy i prvních X časových jednotek.
- `simulate 1 [<=300] {Task(1).Running, Task(2).Running}` – pomocí tohoto výrazu můžeme sledovat, kdy běží úlohy 1 a 2.
- `Pr [<=X] (<>Task(1).Error||Task(2).Error)` – ověřuje pravděpodobnost, že se úloha 1 nebo 2 dostane během X časových jednotek do stavu `Error`.

Simulační dotazy slouží pro vytváření grafů, které znázorňují průběh jednotlivých proměnných nebo kdy se systém nachází v nějakém stavu. Především pomocí nich budeme sledovat, kdy se úlohy nachází v stavu `Running`.

Některé sady nejsou plánovatelné vůbec, ale v některých sadách dojde k překročení časových mezí u nějaké z úloh pouze v okrajových případech. Proto budeme využívat pravděpodobnostní dotazy, abychom mohli zjistit, jak často k porušení mezí dochází. Dále budou využity při ověřování plánovatelnosti sad s ohledem na přerušení, kde nelze využívat symbolické (standardní) dotazy. Výsledek pravděpodobnostního výrazu znamená, že s 99% jistotou můžeme tvrdit, že se úloha dostane do stavu `Error` s touto pravděpodobností.

Kapitola 5

Zhodnocení řešení

Cílem této kapitoly bude analýza plánovatelnosti navržených sad úloh pomocí vytvořeného frameworku. V sekci 5.1 představíme podmínky a scénáře experimentů a poté vyhodnotíme plánovatelnost sad úloh v různých kontextech a podmínkách. Budeme zkoumat, jak se plánovatelnost měnila v závislosti na použitém plánovacím mechanismu a jaké další faktory měly klíčový vliv na výsledek. Výstup budeme zobrazovat v některých případech pomocí Ganttova diagramu. Je důležité vysvětlit, jak jsou jednotlivé stavy reprezentovány. Každý stav je reprezentován barvou. `Error` je reprezentován červenou barvou, `Ready` žlutou, `Stoptask` fialovou, `Running` zelenou a stavy znázorňující ukončení úlohy, `PeriodicTask, SporadicTask, AperiodicTask`, reprezentuje barva modrá.

5.1 Podmínky a scénáře experimentů

5.1.1 Analýza plánovatelnosti na existujících sadách úloh

V úvodní části se zaměříme na ověření plánovatelnosti sad úloh 1-4, které jsou převzaty z literárních zdrojů a upraveny v souladu se specifiky vytvořeného modelu. Tyto sady úloh slouží především jako nástroj k důkladnému ověření validity našeho modelu. Experiment spočívá v porovnání dosažených výsledků s výsledky zveřejněnými v literatuře. Výsledky porovnáme také s výstupy z nástroje Cheddar, jelikož je v něm sady 1-4 možné namodelovat.

Díky dostupnosti těchto sad a jejich výsledků bylo jednodušší identifikovat chyby nebo nesrovnalosti v chování modelu.

5.1.2 Analýza plánovatelnosti na navržených sadách úloh

V tomto případě se budeme zaměřovat na ověření plánovatelnosti sad úloh 5-7, které jsem navrhl. Každá sada úloh obsahuje různé typů úloh a scénářů, přičemž některé sady jsou čistě periodické, jiné aperiodické a nebo kombinují periodické, aperiodické i sporadické úlohy.

Do některých z těchto sad úloh jsou zavedeny nejistoty, a to tím, že nemají striktně deterministické parametry nebo obsahují parametry, které by mohly představovat zdroje nejistoty, jako je čas zavedení úlohy do systému. Čím více nedeterminismu je do úloh zaneseno, tím více stoupá nejistota, složitost modelu a také časová náročnost ověřování vlastností. Nejistotu také může představovat výskyt sporadických úloh. Každá z těchto sad úloh byla pečlivě navržena tak, aby byla plánovatelná pomocí některého z implementovaných plánovacích mechanismů.

5.1.3 Analýza plánovatelnosti sad úloh s přerušeními

Zaměříme se i na zkoumání vlivu přerušení na plánovatelnost sad úloh 5 a 7. Provedeme systematické zkoumání různých scénářů, kde budeme modifikovat dobu mezi příchody přerušení a také dobu trvání samotných přerušení. Tento přístup nám umožní lépe porozumět tomu, jak se plánovatelnost sad úloh mění v reakci na přerušení. Zde budeme ověřovat pouze statistické dotazy. Ověřovat symbolické dotazy není možné, protože pracují s diskrétním časem a délka mezi příchody přerušení a délka trvání přerušení se generuje jako typ double.

5.2 Sady úloh, výsledky a jejich interpretace

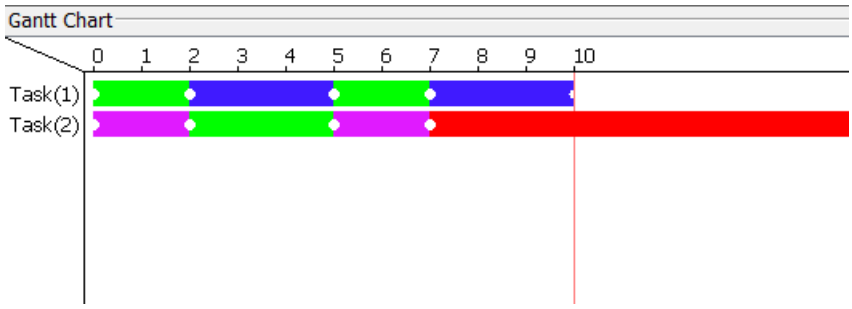
5.2.1 Sada 1

První sada úloh, kterou si představíme, byla převzata z článku [19] a skládá se z dvou periodických úloh. Slouží jako příklad pro ilustrování plánování pomocí mechanismů EDF a LLF.

Úloha	Priorita	In. offset	Min. period	Max. period	Offset	Deadline	BCET	WCET
τ_1	0	0	5	5	0	5	2	2
τ_2	0	0	7	7	0	7	4	4

Tabulka 5.1: Sada 1

Tato sada úloh není plánovatelná ani pomocí jednoho z mechanismů RMA a DMA. Důvodem je, že v čase $t = 7$ dojde k překročení časového meze úlohy τ_2, jak je znázorněno v Ganttově diagramu na obrázku 5.1.



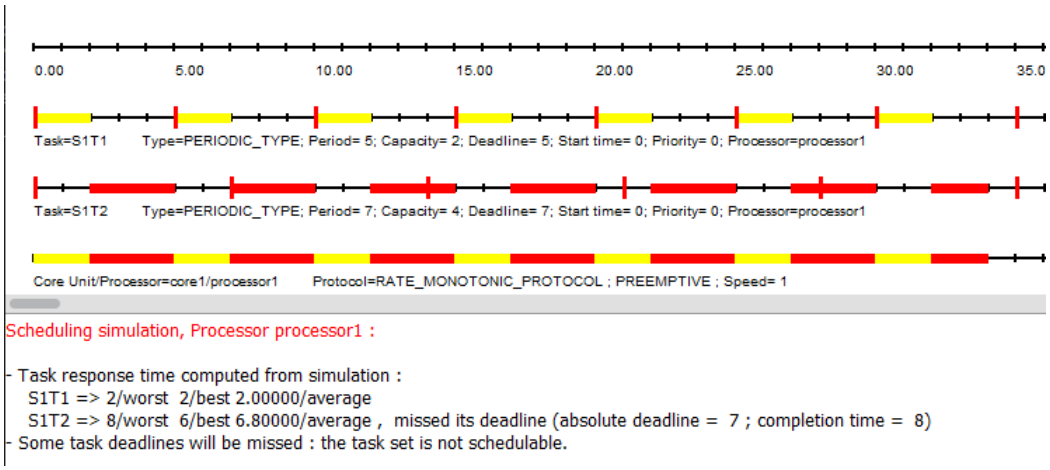
Obrázek 5.1: **Graf znázorňující vstup úlohy τ_2 do Error stavu (RMA, DMA)**

Pokud se ale priorita úloh mění dynamicky, k překročení mezí u žádné z úloh nedojde. V čase $t = 0$ je podle mechanismu EDF úloze τ_1 přiřazena nejvyšší priorita, protože do uplynutí jejího časového limitu zbývá méně času než úloze τ_2. Úloha τ_1 běží bez přerušení po dobu 2 časových jednotek a dokončí se. Následně, když v systému není žádná další úloha, je v čase $t = 2$ spuštěna úloha τ_2, která poběží až do času příchodu nové periody τ_1, tedy do $t = 5$. V tomto okamžiku jsou priority přehodnoceny. Jelikož úloze τ_2 zbývají do uplynutí časového limitu pouze dvě časové jednotky, zatímco úloze τ_1 pět jednotek, je vyšší priorita přiřazena úloze τ_2. Ta dokončí svůj běh (v délce jedné časové jednotky) v čase $t = 6$. [19]

Ganttův diagram na obrázku 5.1 odpovídá stejně jako výsledky v tabulce 5.2 datům zveřejněným v [19]. Výstup z nástroje Cheddar odpovídal získaným výsledkům. Na obrázku 5.2 je znázorněna simulace pro mechanismus RMA.

Mechanismus	Error	Deadlock	Přep. kontextu(t≤35)	Prav. error	Max. 1 běžící úloha
RMA	Ano	Ano	/	[0.990015, 1]	Ano
DMA	Ano	Ano	/	[0.990015, 1]	Ano
EDF	Ne	Ne	13	[0, 0.00998451]	Ano

Tabulka 5.2: Výsledky sada 1



Obrázek 5.2: **Cheddar simulace sada 1 RMA**

5.2.2 Sada 2

Druhá sada úloh byla inspirována sadou zveřejněnou v [4]. Tvoří ji tři periodické úlohy a každá z nich přichází do systému v jiném čase a všechny parametry úloh jsou deterministické. Sada je navržena tak, aby byla plánovatelná pomocí mechanismu RMA, jelikož RMA mechanismus je optimální pro sadu úloh, mezi jejichž parametry platí vztah $D = T$. [19] Všechny úlohy v sadě 1 podmínku splňují.

Úloha	Priorita	Init. offset	Min. period	Max. period	Offset	Deadline	BCET	WCET
τ_1	0	0	5	5	0	5	2	2
τ_2	0	1	4	4	0	4	1	1
τ_3	0	2	20	20	0	20	2	2

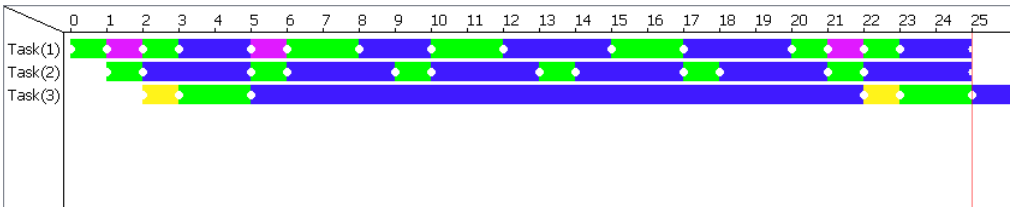
Tabulka 5.3: Sada 2

Plánovatelnost sady jsem na rozdíl od literatury ověřoval také pomocí dalších plánovacích mechanismů. Z výsledků (viz tabulka 5.4) je patrné, že k překročení mezí ani v jednom případě nedojde. Zajímavostí je také to, že při použití mechanismu Round Robin s kvantem větším než 1 produkuje tento mechanismus stejné výsledky jako mechanismus FIFO. Lze to vysvětit tím, že doba provádění všech úloh je menší nebo rovna délce kvanta a také tím, jak jsou zvoleny délky period. Úlohy se tak stihnou dokončit ještě před vypršením kvanta a nedojde k přepnutí kontextu.

Mechanismus	Error	Deadlock	Přep. kontextu(t≤50)	Prav. error	Max. 1 běžící úloha
RMA	Ne	Ne	28	[0, 0.00998451]	Ano
DMA	Ne	Ne	28	[0, 0.00998451]	Ano
EDF	Ne	Ne	27	[0, 0.00998451]	Ano
FIFO	Ne	Ne	24	[0, 0.00998451]	Ano
RR(2)	Ne	Ne	24	[0, 0.00998451]	Ano
RR(1)	Ne	Ne	37	[0, 0.00998451]	Ano

Tabulka 5.4: Sada 2 výsledky

Ganttův diagram pro mechanismus RMA na obrázku 5.3 odpovídá plánu v literatuře. Výsledky jsem také ověřoval v nástroji Cheddar a dosáhl stejných výsledků, kdy u žádného z mechanismů nedošlo k porušení časových mezí.



Obrázek 5.3: **Ilustrace průběhu sady 2 (RMA)**

5.2.3 Sada 3

Další sadu, pomocí které budeme ověřovat správnost našeho modelu, je sada 4 aperiodických úloh. Jedná se o sadu z knihy [4]. Úlohy τ_1 a τ_2 přichází v čase $t = 0$, úloha τ_3 v čase $t = 2$ a τ_4 v čase $t = 5$.

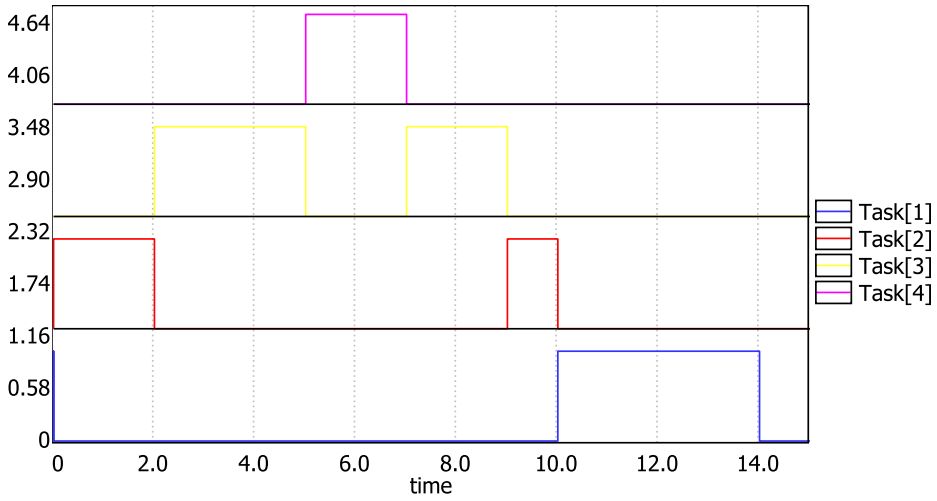
Úloha	Priorita	Init. offset	Min. period	Max. period	Offset	Deadline	BCET	WCET
τ_1	0	0	0	0	0	15	4	4
τ_2	0	0	0	0	0	12	3	3
τ_3	0	2	0	0	0	7	5	5
τ_4	0	5	0	0	0	3	2	2

Tabulka 5.5: Sada 3

Bylo dosaženo stejných výsledků jako v literatuře (viz 5.6). Výsledky se také shodovaly s výstupem nástroje Cheddar. Sada není pomocí FIFO mechanismu plánovatelná, jelikož dojde k překročení meze u úlohy τ_4 v čase $t = 8$ a úloha τ_3 překročí svou mez v čase $t \geq 9$. Pomocí EDF mechanismu jsme plánovatelnosti dosáhli. Analýzu jsme také rozšířili o mechanismus DMA, který produkuje totožný plán (viz 5.4) jako EDF. Použití mechanismu RMA v tomto případě není možné, jelikož se nejedná o periodickou sadu.

Mechanismus	Error	Úlohy dok.	Přep. kontextu(t≤15)	Pr. error	1 běžící úloha
DMA	Ne	Ano	6	[0, 0.00998451]	Ano
EDF	Ne	Ano	5	[0, 0.00998451]	Ano
FIFO	Ano	Ne	3	[0.990015, 1]	Ano

Tabulka 5.6: Sada 3 výsledky



Obrázek 5.4: **Ilustrace průběhu sady 3 (DMA, EDF)**

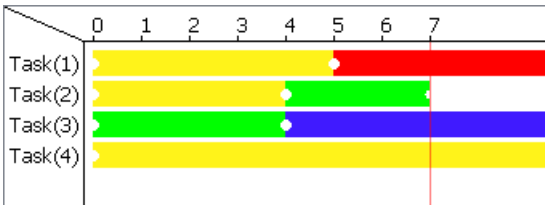
5.2.4 Sada 4

Tato sada byla stejně jako sada 1 inspirována sadou zveřejněnou v [19]. Tvoří ji 4 periodické úlohy pro jejichž parametry D a T platí, že $D \leq T$. Na základě statické analýzy této sady můžeme říci, že nejspíš nebude plánovatelná pomocí RMA, ale pomocí DMA ano, protože RMA je optimální pro úlohy, kde se rovnají parametry D a T [19], což u úloh τ_1 a τ_2 splněno není. DMA je optimální pro sady úloh mezi jejichž parametry platí vztah $D \leq T$ [19].

Úloha	Priorita	Init. offset	Min. period	Max. period	Offset	Deadline	BCET	WCET
τ_1	0	0	20	20	0	5	3	3
τ_2	0	0	12	12	0	7	3	3
τ_3	0	0	10	10	0	10	4	4
τ_4	0	0	20	20	0	20	3	3

Tabulka 5.7: Sada 4

Dosáhl jsem stejných výsledků jako v literatuře. Jak jsem předpokládal, sada není plánovatelná pomocí mechanismu RMA, jelikož v čase 5 dojde k překročení časové meze úlohy τ_1, jak znázorňuje Ganttův diagram na obrázku 5.5. DMA vytváří přípustný plán a k překročení časových mezí nedojde.



Obrázek 5.5: **Ganntův diagram RMA**

Analýzu jsem rozšířil o další plánovací mechanismy, a to konkrétně o EDF, FIFO a Round Robin s kvantem 1 a 3. Plánovatelnost sady byl schopen zajistit pouze mechanismus EDF a v porovnání s mechanismem DMA došlo k menšímu počtu přepnutí kontextu. U ostatních mechanismů došlo stejně jako RMA k překročení meze úlohy τ_1 v čase $t \in (5, 6]$. Totožné výsledky produkoval také nástroj Cheddar, kterým jsme ověřovali plánovatelnost pomocí všech mechanismů kromě RMA. Ten nebylo možné použít, jelikož jeho použití je podmíněno rovnosti parametrů D a T, což je u úlohy τ_3 porušeno.

Mechanismus	**Error**	**Deadlock**	**Přep. kontextu(t≤50)**	**Pr. error**	**1 běžící úloha**
DMA	Ne	Ne	25	[0, 0.00998451]	Ano
EDF	Ne	Ne	20	[0, 0.00998451]	Ano
FIFO	Ano	Ano	/	[0.990015, 1]	Ano
RR(1)	Ano	Ano	/	/	Ano
RR(3)	Ano	Ano	/	/	Ano
RMA	Ano	Ne	/	[0.990015, 1]	Ano

Tabulka 5.8: Sada 4 výsledky

5.2.5 Sada 5

Sada obsahuje 3 periodické úlohy. V této sadě se setkáváme s nedeterministickými časy dokončení úloh. To znamená, že nevíme přesně, kdy každá úloha skončí, což může mít vliv na plánovatelnost. Víme pouze, že k dokončení dojde v intervalu $t \in [BCET, WCET]$. Sadu jsem analyzoval pomocí všech implementovaných mechanismů.

Úloha	**Priorita**	**Init. offset**	**Min. period**	**Max. period**	**Offset**	**Deadline**	**BCET**	**WCET**
τ_1	2	1	50	50	2	10	2	3
τ_2	1	4	60	60	1	7	1	3
τ_3	2	1	45	45	3	6	1	2

Tabulka 5.9: Sada 5

Tabulka 5.10 obsahuje souhrn výsledků. Sada je plánovatelná pomocí mechanismů EDF, DMA, RMA a Round Robin s kvantem 2. V ostatních případech dochází k překročení časové meze u úlohy τ_3 v čase $t \in (6, 7]$. Na základě výsledku pravděpodobnostních dotazů lze tvrdit, že k překročení mezí dojde u mechanismů RMA a FIFO zhruba v polovině případů. Nejedná se tedy o situaci, která nastane vždy.

Mechanismus	**Error**	**Deadlock**	**Přep. kontextu(t≤50)**	**Pr. error**	**1 běžící úloha**
DMA	Ne	Ne	5	[0, 0.00998451]	Ano
EDF	Ne	Ne	5	[0, 0.00998451]	Ano
FIFO	Ano	/	/	[0.489192, 0.509191]	Ano
RR(2)	Ne	Ne	5	/	Ano
FPS	Ano	/	0	[0.495663, 0.51566]	Ano
RMA	Ne	Ne	5	[0, 0.00998451]	Ano

Tabulka 5.10: Sada 5 výsledky

Výsledky s přerušeními

V této části si zanalyzujeme vliv přerušení na plánovatelnost sady 5. Intervaly mezi příchody přerušení a doby obsluhy přerušení jsou znázorněny v tabulce 5.11. Intervaly v experimentu 1 byly navrženy, aby byla sada i s přerušeními s velkou pravděpodobností plánovatelná. U experimentů 2 a 3 jsem zvyšoval frekvenci mezi příchody přerušení a u experimentů 3 a 4 jsem zvyšoval dobu obsluhy přerušení.

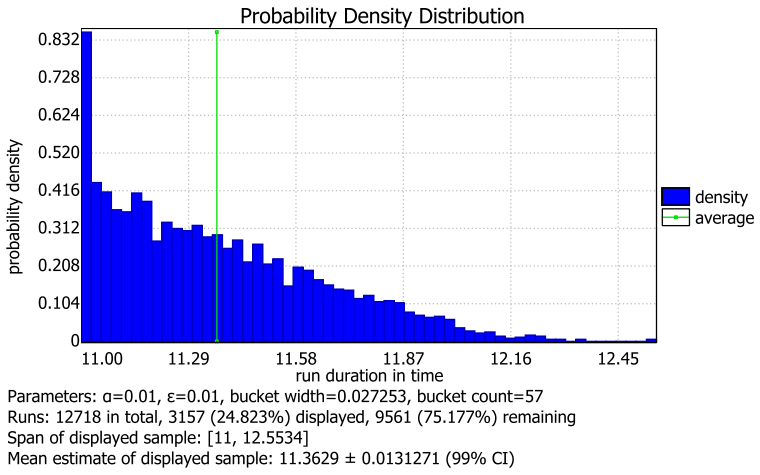
	Interval mezi příchody přerušení	**Interval doby obsluhy přerušení**
Experiment 1	[8.0, 10.0)	[0.1, 0.5)
Experiment 2	[4.0, 5.0)	[0.1, 0.5)
Experiment 3	[1.5, 2.0)	[0.1, 0.5)
Experiment 4	[8.0, 10.0)	[0.5, 0.8)
Experiment 5	[8.0, 10.0)	[0.8, 1.0)

Tabulka 5.11: Sada 5 experimenty scénáře

Výsledky v tabulce 5.12 znázorňují pravděpodobnost, že se sada při použití jednoho z mechanismů dostane do stavu `Error` během prvních 200 kroků simulace. Pravděpodobnosti jsou uvedeny pro každý z experimentů. Z výsledků je patrné, že na plánovatelnost má u mnou provedených experimentů větší dopad zvyšování frekvence příchodu přerušení než zvyšování délky obsluhy přerušení. Ve všech experimentech, kromě experimentu 5, byla nejmenší pravděpodobnost, že dojde k překročení mezí při použití mechanismu EDF.

	EDF	**RMA**	**DMA**
Experiment 1	[0.000788202, 0.0146731]	[0.000788202, 0.0146731]	[0.000788202, 0.0146731]
Experiment 2	[0.0644374, 0.0836153]	[0.0572148, 0.0763039]	[0.0640095, 0.0831845]
Experiment 3	[0.23909, 0.258901]	[0.232959, 0.252762]	[0.238421, 0.25823]
Experiment 4	[0.0148453, 0.0324876]	[0.0205155, 0.0385701]	[0.0120223, 0.0293759]
Experiment 5	[0.0579237, 0.0770239]	[0.0475622, 0.0664997]	[0.0518603, 0.0708735]

Tabulka 5.12: Sada 5 přerušení výsledky



Obrázek 5.6: **Hustota rozložení pravděpodobnosti, DMA (Experiment 3)**

5.2.6 Sada 6

Sada 6 obsahuje periodickou, sporadickou a aperiodickou úlohu. Na této sadě není možné provádět statistické dotazy, jelikož stav `SporadicTask` má nespecifikovaný invariant, což ale správně odpovídá chování sporadické úlohy. To nám jen potvrzuje, jak velkou nejistotou jsou sporadické úlohy a jak je ověřování sad se sporadickými úlohami náročné.

Úloha	Priorita	Init. offset	Min. period	Max. period	Offset	Deadline	BCET	WCET
τ_1	0	7	18	18	0	18	7	7
τ_2	0	20	0	0	0	4	2	2
τ_3	0	2	20	0	0	20	5	5

Tabulka 5.13: Sada 6

U této sady nebylo ani u jednoho z použitých mechanismů možné ověřit, zda nedojde k uváznutí. Je to způsobeno právě sporadickou úlohou, která obsahuje pouze minimální periodu a neobsahuje žádnou horní hranici, což způsobuje obrovský nárůst stavového prostoru, jehož prohledávání je tak časově náročné. U mechanismu EDF nebylo možné ani po několika hodinách ověřit, zda se nedostane nějaká z úloh do chybového stavu. Příčinu této konkrétní situace způsobuje nejspíše rozdíl v implementaci mechanismu EDF oproti ostatním použitým mechanismům, u kterých se mi tuto vlastnost ověřit podařilo.

Mechanismus	Error	1 běžící úloha
DMA	Ne	Ano
FIFO	Ne	Ano
RR(2)	Ne	Ano
RR(5)	Ne	Ano

Tabulka 5.14: Sada 6 výsledky

5.2.7 Sada 7

Další navržená sada obsahuje tři aperiodické úlohy a jednu periodickou úlohu. Nedeterminismus parametrů je pouze u úloh τ_1 a τ_2. První má nedeterministickou dobu provádění a druhá nedeterministickou délku periody.

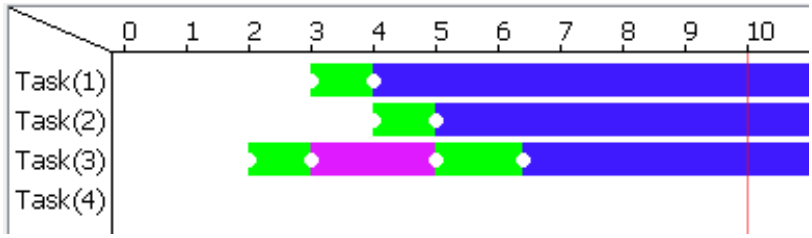
Úloha	Priorita	Init. offset	Min. period	Max. period	Offset	Deadline	BCET	WCET
τ_1	2	2	0	0	1	8	1	3
τ_2	4	4	17	19	0	2	1	1
τ_3	1	0	0	0	2	7	2	3
τ_4	0	10	0	0	0	8	4	4

Tabulka 5.15: Sada 7

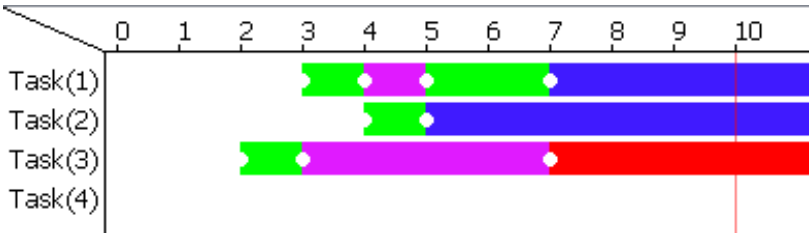
Přípustný plán vytvořily pouze mechanismy EDF a DMA. U mechanismů FIFO a Round Robin došlo k překročení časové meze u úlohy τ_2 v čase $t = 6$. Porušení časové meze nastalo při použití mechanismu FPS v čase $t = 7$ u úlohy τ_3. U FPS ale k této situaci dochází s menší pravděpodobností než u Round Robin a FIFO, kde k tomu dojde skoro jistě. Obě situace, které mohou nastat u FPS jsou znázorněny na obrázku 5.7.

Mechanismus	Error	Deadlock	Přep. kontextu(t≤50)	Pr. error	1 běžící úloha
DMA	Ne	Ne	7	[0, 0.00998451]	Ano
EDF	Ne	Ne	7	[0, 0.00998451]	Ano
FIFO	Ano	Ano	/	[0.990015, 1]	Ano
RR(4)	Ano	Ano	/	[0.990015, 1]	Ano
FPS	Ano	Ano	/	[0.737917, 0.757732]	Ano

Tabulka 5.16: Sada 7 výsledky



(a) Přípustný plán



(b) Nepřípustný plán

Obrázek 5.7: **Sada 7 FPS**

Výsledky s přerušeními

Tabulka 5.17 obsahuje podobně jako u sady 5 scénáře experimentů. U scénářů 2 a 3 jsem modifikoval oproti scénáři 1 interval příchodu přerušení. Na základě výsledků jsem pak ve scénářích 4 a 5 pracoval s intervalem příchodu přerušení použitém v scénáři 3 a modifikoval interval doby obsluhy přerušení.

	Interval příchodu přerušení	Interval doby obsluhy přerušení
Experiment 1	[9.3, 12.5)	[0.5, 0.8)
Experiment 2	[6.2, 8.3)	[0.5, 0.8)
Experiment 3	[3.4, 4.5)	[0.5, 0.8)
Experiment 4	[3.4, 4.5)	[0.8, 1.2)
Experiment 5	[3.4, 4.5)	[0.75, 1.0)

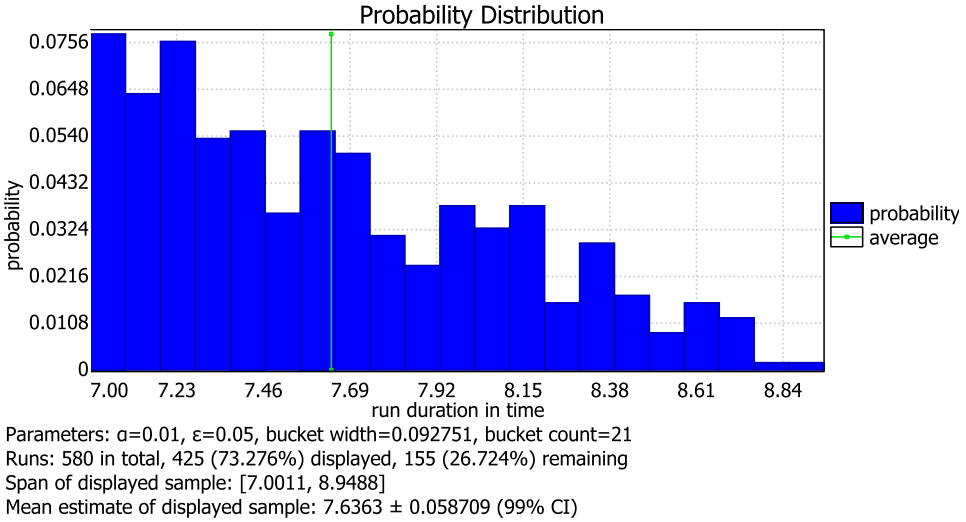
Tabulka 5.17: Sada 7 scénáře

V experimentech 1 a 2 nezpůsobily při použití mechanismů EDF a DMA přerušení nárůst pravděpodobnosti, u žádného z běhů nedošlo k porušení časových mezí. U těchto mechanismů jsme zaznamenali porušení mezí až od 3. experimentu. V tomto scénáři ale dochází k porušení mezí velmi zřídka. Rozhodl jsem se v posledních dvou experimentech tento interval neměnit a upravoval jsem dobu obsluhy přerušení. Rozdíly v pravděpodobnosti experimentů 4 a 5 jsou markantní. Pokud obsluha přerušení trvá déle než jednu časovou jednotku, dochází k porušení časových mezí přibližně s pravděpodobností 0.83 (experiment 4). Pokud ale interval zmenšíme jako v posledním experimentu na [0.75, 1.0) a doba obsluhy přerušení tak nepřesáhne 1, pravděpodobnost porušení mezí výrazným způsobem klesá k hodnotě kolem 0.12.

	EDF	**FPS**	**DMA**
Experiment 1	[0, 0.00998451]	[0.735683, 0.7555]	[0, 0.00998451]
Experiment 2	[0, 0.00998451]	[0.794266, 0.813996]	[0, 0.00998451]
Experiment 3	[0.0137475, 0.0312851]	[0.949832, 0.968381]	[0.017677, 0.0355469]
Experiment 4	[0.822438, 0.842107]	[0.988449, 0.999992]	[0.830232, 0.849882]
Experiment 5	[0.122489, 0.142049]	[0.984495, 0.998811]	[0.118912, 0.138458]

Tabulka 5.18: Sada 7 přerušení výsledky

V prvních dvou experimentech byl nárůst pravděpodobnosti při použití mechanismu FPS v řádech setin. U experimentů 3-5 ale přesahovala pravděpodobnost 0.9 a k překročení mezí došlo takřka v každém běhu simulace.



Obrázek 5.8: **Funkce rozložení pravděpodobnosti, FPS (Experiment 1)**

Kapitola 6

Závěr

Cílem této bakalářské práce bylo seznámení se se systémy pracujícími v reálném čase, nejistotami a plánovacími mechanismy, souvisejícími s těmito systémy, statistickým ověřováním modelů, dále to byly návrh a implementace přístupu k analýze plánovatelnosti s ohledem na nejistoty, vytvoření vhodných sad úloh reálného času a ověření jejich plánovatelnosti. Všechny tyto cíle byly splněny.

Prvním krokem bylo provedení rešerše ve zmíněných oblastech. Na základě rešerše a konzultace s vedoucím práce jsem se rozhodl pro vytvoření simulačního modelu v nástroji UPPAAL, který zahrnuje modely plánovacích mechanismů, úlohy, plánovače a přerušovacího mechanismu. Jako nejistotu, která bude do sad úloh jsem zvolil zavedení přerušení do systému a také možnost zvolit nedeterministické parametry u jednotlivých úloh. Dále je možné určit typ úlohy, kdy především úlohy sporadické představují pro RT systémy velkou nejistotu. Některé ze sad úloh byly převzaty z dostupné literatury a simulace provedené nad těmito modely sloužily především pro validaci vytvořeného modelu. U těchto sad došlo také k porovnání získaných výsledků také s výstupem nástroje Cheddar. Pro zbývající sady byly vytvořeny dva scénáře experimentů. V prvním případě byla ověřována jejich plánovatelnost a další zajímavé parametry bez přerušení. V druhém případě jsme ověřovali tyto vlastnosti v případě, že jsou do systému zanesena přerušení.

Implementovaný model umožňuje jednoduchým způsobem ověřit plánovatelnost jednoduchých i složitějších sad úloh. Dále je také možné na základě získaných statistických informací o úlohách například upravit parametry jednotlivých úloh pro efektivnější využití zdrojů.

Výslednou práci by bylo možné rozšířit o další plánovací mechanismy, zavést závislost úloh, rozšířit parametry úloh nebo některé parametry učinit nedeterministické. Dalším možným rozšířením této práce by mohla být implementace víceprocesorového nebo vícefrontového plánování. Zajímavým rozšířením a také přiblížení chování modelu k realitě by bylo přidání modelu procesoru.

Literatura

[1] Atlas, A. a Bestavros, A. Statistical rate monotonic scheduling. In: *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No.98CB36279)*. Los Alamitos CA: IEEE, 1998, s. 123–132. ISBN 9780818692123.

[2] Behrmann, G., David, A. a Larsen, K. G. A Tutorial on Uppaal. In: *Formal Methods for the Design of Real-Time Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, s. 200–236. ISBN 9783540230687.

[3] Buttazzo, G., Lipari, G. a Abeni, L. Elastic task model for adaptive rate control. In: *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No.98CB36279)*. Los Alamitos CA: IEEE, 1998, s. 286–295. ISBN 9780818692123.

[4] Cheng, A. M. K. *Real-time systems: scheduling, analysis, and verification*. 1. vyd. Newark: WILEY, 2003. ISBN 9780471460848.

[5] Cottet, F. *Scheduling in real-time systems*. Chichester: John Wiley, 2002. ISBN 0-470-84766-2.

[6] David, A., Larsen, K. G., Legay, A., Mikučionis, M. a Poulsen, D. B. Uppaal SMC tutorial. *International journal on software tools for technology transfer*. Berlin/Heidelberg: Springer Berlin Heidelberg. 2015, sv. 17, č. 4, s. 397–415. ISSN 1433-2779.

[7] Hartmanns, A. a Hermanns, H. In the quantitative automata zoo. *Science of computer programming*. Elsevier B.V. 2015, sv. 112, s. 3–23. ISSN 0167-6423.

[8] Hubbard, D. W. *How to measure anything : finding the value of "intangibles"in business*. 2nd ed. Hoboken: John Wiley & Sons, 2010. ISBN 978-0-470-53939-2.

[9] Kopetz, H. *Real-time systems : design principles for distributed embedded applications*. 2nd ed. Boston: Springer, 2011. Real-time systems series. ISBN 978-1-4419-8236-0.

[10] Laplante, P. *The certainty of uncertainty in real-time systems*. New York, NY: IEEE, 2004. ISSN 1094-6969.

[11] Malý, M. *Hradla, volty, jednočipy : úvod do bastlení*. 1. vydání. Praha: CZ.NIC, z.s.p.o., 2017. CZ.NIC. ISBN 978-80-88168-23-2.

[12] Mok, A. a Chen, D. A multiframe model for real-time tasks. *IEEE transactions on software engineering*. New York, NY: IEEE. 1997, sv. 23, č. 10, s. 635–645. ISSN 0098-5589.

[13] Perathoner, S., Wandeler, E., Thiele, L., Hamann, A., Schliecker, S. et al. Influence of different abstractions on the performance analysis of distributed hard real-time systems. *Design automation for embedded systems.* Boston: Springer US. 2009, sv. 13, 1-2, s. 27–49. ISSN 0929-5585.

[14] Pilát, M. *Modální logika* [online]. 2022 [cit. 2024-03-25]. Dostupné z: https://martinpilat.com/cs/multiagentni-systemy/modalni-logika.

[15] Pinedo, M. L. *Scheduling : theory, algorithms, and systems.* 4th ed. New York: Springer, 2012. ISBN 978-1-4614-1986-0.

[16] Shan, L., Graf, S., Quinton, S. a Fejoz, L. A Framework for Evaluating Schedulability Analysis Tools. 1st ed. Cham: Springer International Publishing. 2017, s. 539–559. ISSN 0302-9743.

[17] Singhoff, F. a Tran, H. N. *Cheddar – an open-source real-time schedulability tool/scheduling simulator* [online]. October 2002 [cit. 2024-03-05]. Dostupné z: http://beru.univ-brest.fr/cheddar/.

[18] Strnadel, J. Návrh časově kritických systémů II: úlohy reálného času. *Automa.* 1. vydání. FCC Public. Prosinec 2010, sv. 1, s. 18–19. ISSN 1210-9592. Dostupné z: https://automa.cz/Aton/FileRepository/pdf_articles/42455.pdf.

[19] Strnadel, J. Návrh časově kritických systémů III: priorita úloh. *Automa.* 1. vydání. FCC Public. Únor 2011, sv. 1, s. 50–52. ISSN 1210-9592. Dostupné z: http://www.odbornecasopisy.cz/res/pdf/42988.pdf.

[20] Strnadel, J. Návrh časově kritických systémů I: specifikace a verifikace. *Automa.* 1. vydání. FCC Public. Říjen 2010, sv. 1, s. 42–44. ISSN 1210-9592. Dostupné z: https://automa.cz/Aton/FileRepository/pdf_articles/42105.pdf.

[21] Uppsala University, I. department. *Times* [online]. November 2008. Dostupné z: http://www.timestool.com/.

[22] Uppsala University, I. department. *Uppaal* [online]. April 2024. Dostupné z: https://uppaal.org/.

Příloha A

Obsah paměťového média

Příloha popisuje obsah přiloženého paměťového média. V kořenovém adresáři se nachází 3 podadresáře. Složka `src` obsahuje model implementovaný nástrojem UPPAAL ve formátu `xml` a model implementovaný v nástroji Cheddar ve formátu `xml`. Složka `doc` obsahuje technickou zprávu ve formátu `pdf`, zdrojové kódy ve formátu LaTeX a podadresář `obrazky-figures`, který obsahuje obrázky použité v práci. Složka `manual` obsahuje soubor ve formátu `pdf` popisující návod k práci s vytvořeným frameworkem.