



evropský
sociální
fond v ČR



EVROPSKÁ UNIE



MINISTERSTVO ŠKOLSTVÍ,
MLÁDEŽE A TĚLOVÝCHOVY



OP Vzdělávání
pro konkurenčeschopnost



UNIVERSITAS
OSTRAVIENSIS

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

MODELOVÁNÍ A SIMULACE

**URČENO PRO VZDĚLÁVÁNÍ V AKREDITOVANÝCH
STUDIJNÍCH PROGRAMECH**

**MICHAL JANOŠEK
RADIM FARANA**

ČÍSLO OPERAČNÍHO PROGRAMU: CZ.1.07

NÁZEV OPERAČNÍHO PROGRAMU:

VZDĚLÁVÁNÍ PRO KONKURENCESCHOPNOST

OPATŘENÍ: 7.2

ČÍSLO OBLASTI PODPORY: 7.2.2

**INOVACE VÝUKY INFORMATICKÝCH PŘEDMĚTŮ VE
STUDIJNÍCH PROGRAMECH OSTRAVSKÉ UNIVERZITY**

REGISTRACNÍ ČÍSLO PROJEKTU: CZ.1.07/2.2.00/28.0245

OSTRAVA 2016

Tento projekt je spolufinancován Evropským sociálním fondem a státním rozpočtem České republiky

Recenzent: prof. RNDr. Ing. Ivan Křivý, CSc.

Název: Modelování a simulace
Autoři: RNDr. Michal Janošek, Ph.D.
prof. Ing. Radim Farana, CSc.
Vydání: čtvrté, 2016
Počet stran: 158

Jazyková korektura nebyla provedena, za jazykovou stránku odpovídají autoři.

© Michal Janošek, Radim Farana
© Ostravská univerzita v Ostravě

OBSAH

1 ÚVOD A PRŮVODCE STUDIEM	7
1.1 O modelování a simulaci	7
1.2 Průvodce textem.....	8
2 ZÁKLADNÍ POJMY.....	11
2.1 Systém.....	11
2.2 Model	16
2.3 Modelování	19
2.4 Simulace.....	21
2.5 Simulace na počítačích	23
2.6 Termíny používané při číslicové simulaci	24
3 KOMPLEXNÍ SYSTÉMY	27
3.1 Komplexní systémy	27
3.2 Jednoduché a složité systémy	29
3.3 Charakteristiky komplexních systémů.....	30
3.4 Studium komplexních systémů	32
3.5 Zpětná vazba	33
4 MATEMATICKÉ PROSTŘEDKY PRO MODELOVÁNÍ A SIMULACI.....	39
4.1 Matematické prostředky.....	39
4.2 Matematické metody.....	42
4.3 Základní fáze simulace	49
5 ORGANIZACE SIMULAČNÍHO MODELU	53
5.1 Hlavní úkoly při konstrukci simulačního jádra.....	53
5.2 Zobrazení stavů simulovaného systému	54
5.3 Zobrazení stavových změn	54
5.4 Zobrazení času	54
5.5 Synchronizace výpočtu	55
5.6 Vstupy a výstupy simulačních programů.....	56
6 ZÁKLADY TEORIE DISKRÉTNÍCH DYNAMICKÝCH SYSTÉMŮ	59
6.1 Definice diskrétního dynamického systému	59
6.2 Stacionární trajektorie diskrétního dynamického systému	61
6.3 Analýza konkrétního diskrétního dynamického systému	63
7 ALGORITMIZACE DISKRÉTNÍCH SIMULAČNÍCH MODELŮ	67
7.1 Procesy	67
7.2 Vnější stavy procesů	68
7.3 Přechody mezi vnějšími stavami procesů	69
7.4 Vnitřní stavы procesů	71
7.5 Kalendáře událostí a jejich základní funkce	71
7.6 Implementace kalendáře událostí.....	72
7.7 Hierarchické kalendáře událostí.....	74
7.8 Plánování událostí	74

8 GENEROVÁNÍ PSEUDONÁHODNÝCH ČÍSEL	77
8.1 Algoritmy pro generování pseudonáhodných čísel.....	79
8.2 Generování pseudonáh. čísel z daného rozdělení	80
8.3 Testování generátoru.....	81
8.4 Implementace generátoru pseudonáhodných čísel.....	82
9 ZÁKLADY TEORIE SPOJITÝCH DYNAMICKÝCH SYSTÉMŮ	83
9.1 Definice spojitého dynamického systému a jeho řešení	83
9.2 Existence a jednoznačnost řešení spojitého dynamického systému	85
9.3 Stabilita řešení spojitého dynamického systému	85
9.4 Stabilita řešení lineárních dynamických systémů	88
9.5 Stabilita řešení nelineárních dynamických systémů	91
10 ALGORITMIZACE SPOJITÝCH SIMULAČNÍCH MODELŮ	95
10.1 Základní pojmy	96
10.2 Metody Rungeho-Kutty	97
10.3 Víceuzlové metody	99
10.4 Metody pro řešení tuhých (stiff) soustav	101
10.5 Posouzení metod numerického řešení soustav obyčejných dif. rovnic	102
11 MODELY HROMADNÉ OBSLUHY	105
11.1 Úvod.....	105
11.2 Klasifikace systémů hromadné obsluhy.....	108
11.3 Simulace systémů hromadné obsluhy v software Witness 2007	109
11.3.1 Pojmový model simulovaného systému hromadné obsluhy .	110
11.3.2 Tvorba simulačního modelu.....	110
11.3.3 Nastavení součásti	112
11.3.4 Nastavení zásobníku	114
11.3.5 Nastavení stroje	114
11.3.6 Definování vazeb mezi jednotlivými prvky modelu.....	116
11.3.7 Spuštění a ovládání simulačního pokusu	118
11.3.8 Automatické výstupy ze simulačního pokusu	119
11.3.9 Odhady požadovaných provozních charakteristik systému .	123
11.3.10 Závěr	125
11.4 Příklad modelu systému hromadné obsluhy v jazyku GPSS	125
11.5 Popis modelu v jazyku SIMULA.....	127
12 PROGRAMOVACÍ PROSTŘEDKY PRO MODELOVÁNÍ A SIMULACI	131
12.1 Rozbor možností	131
12.2 Simulační programovací jazyky	133
12.3 Základní klasifikace simulačních jazyků	135
12.4 Jazyky základního typu A	136
12.5 Jazyky základního typu AT	138
12.6 Jazyky základního typu T	141
12.7 Jazyky s elementárními prvky	142
12.8 Poznámka ke kombinované simulaci.....	144

13 ORGANIZACE SIMULAČNÍ STUDIE, METODY JEJÍHO VYHODNOCENÍ.....	151
13.1 Řízení simulační studie	151
LITERATURA.....	157

Vysvětlivky k používaným symbolům



Průvodce studiem – vstup autora do textu, specifický způsob, kterým se studentem komunikuje, povzbuzuje jej, doplňuje text o další informace



Příklad – objasnění nebo konkretizování problematiky na příkladu ze života, z praxe, ze společenské reality, apod.



Pojmy k zapamatování.



Shrnutí – shrnutí předcházející látky, shrnutí kapitoly.



Literatura – použitá ve studijním materiálu, pro doplnění a rozšíření poznatků.



Kontrolní otázky a úkoly – prověřují, do jaké míry studující text a problematiku pochopil, zapamatoval si podstatné a důležité informace a zda je dokáže aplikovat při řešení problémů.



Úkoly k textu – je potřeba je splnit neprodleně, neboť pomáhají dobrému zvládnutí následující látky.



Korespondenční úkoly – při jejich plnění postupuje studující podle pokynů s notnou dávkou vlastní iniciativy. Úkoly se průběžně evidují a hodnotí v průběhu celého kurzu.



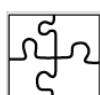
Úkoly k zamýšlení.



Část pro zájemce – přináší látku a úkoly rozšiřující úroveň základního kurzu. Pasáže a úkoly jsou dobrovolné.



Testy a otázky – ke kterým řešení, odpovědi a výsledky studující najdou v rámci studijní opory.



Řešení a odpovědi – vážou se na konkrétní úkoly, zadání a testy.

1 Úvod a průvodce studiem

V této kapitole se dozvítíte:

- Informace o překládané opoře
- Obsah této opory

Po prostudování této opory byste měli:

- získat přehled o matematických prostředcích a metodách vhodných pro modelování a simulaci,
- získat základní informace o teorii diskrétních i spojitéch dynamických systémů a naučíte se správně používat příslušnou odbornou terminologii,
- pochopit základní principy algoritmizace jak diskrétních, tak i spojitéh simulačních modelů,
- poznat nejvýznamnější oblasti aplikace modelování a simulace,
- naučit se vytvářet jednodušší simulační modely a experimentovat s nimi.

Klíčová slova této kapitoly:

Úvod, průvodce studiem, obsah.

Doba potřebná ke studiu: 1 hodina

Průvodce studiem

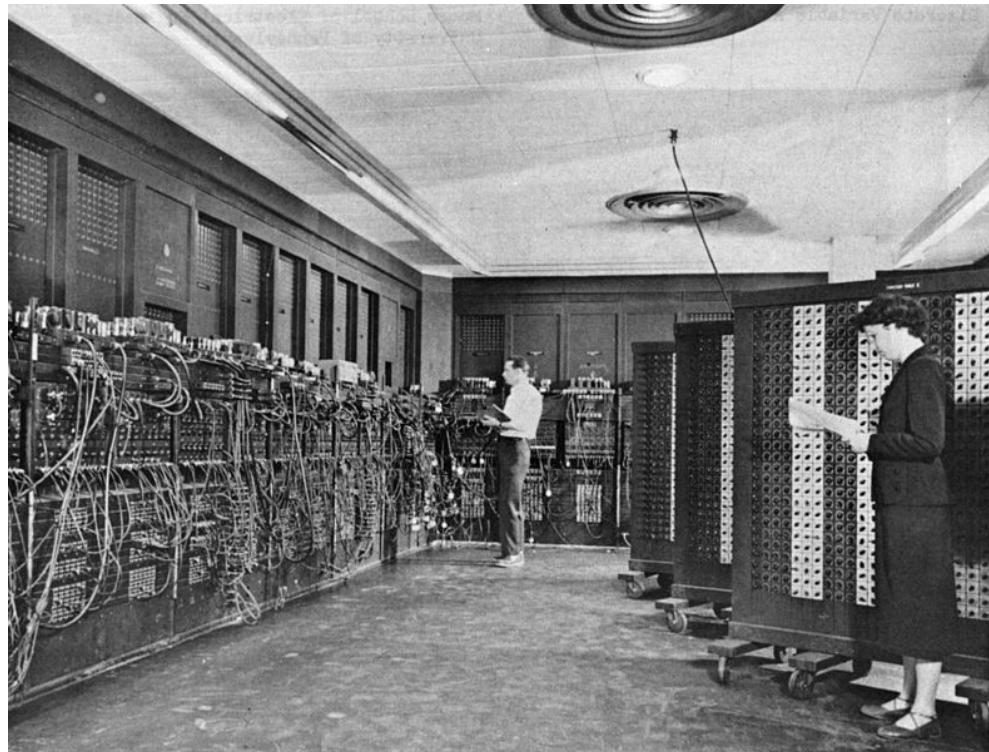
Předkládaná opora je určena pro jednosemestrální studium modelování a simulace. Pokrývá v podstatě učivo přednášené v současnosti v kurzech MOSIM prezenčního studia a XMOSM kombinovaného studia informatiky, oboru informační systémy. Tato opora vznikla sloučením původně dvou opor pro kurzy MOSI1 (Křivý a Kindler 2003a) a MOSI2 (Křivý a Kindler 2003b) a vybírá z těchto opor důležité pasáže.



1.1 O modelování a simulaci

Kdybychom chtěli hledat počátky modelování a simulace, těžko bychom mohli stanovit nějaké datum. Člověk pracoval s modely už od počátku svého bytí, kdy začal vnímat své okolí. Lidský mozek (a nejen ten lidský) pracuje s tzv. mentálními modely, prostřednictvím nichž vnímá okolní realitu. Těžko si představit, že by bylo možné vnímat okolní realitu úplně se vším, co k ní patří. Díváme-li se na dům, automaticky si vytváříme v mysli jeho model. Tak stejně to je se simulací. Pokud si v mysli přehraváme různé scénáře, které mohou v budoucnu nastat, jedná se zjednodušeně řečeno o simulaci.

Lépe je v tomto případě vhodné hovořit o počítačové simulaci. Tak přišla na svět spolu s rozvojem elektronických počítačů na počátku 40. let 20. století. Jedna z prvních počítačových simulací byla použita během druhé světové války v rámci projektu Manhattan a to k simulaci jaderného výbuchu. Velmi známým je také počítač ENIAC, představený roku 1946, který byl prvním obecně použitelným počítačem, jelikož mohl být přrogramován a schopen řešit různé úlohy (Obrázek 1).



Obrázek 1 – Počítač ENIAC (U.S.Army)

S využitím počítačové simulace na výkonných počítačích tak můžeme simulovat procesy, které bychom jinak obtížně pozorovali, byli schopni zkoumat, nebo by byly nákladné realizovat v reálném životě. Může tak jít o simulaci jevů trvající miliony let, jevů na mikroskopické úrovni, či nahradu fyzických modelů, které by jinak byly velmi nákladné.

1.2 Průvodce textem

Celý modul je rozdělen do 13 lekcí, u nichž jsou dodrženy následující pravidla:



- je specifikován cíl lekce (tedy to, co by měl student po jejím prostudování umět, znát, pochopit),
- vlastní výklad učiva, popř. otázky k textu,
- řešené příklady
- kontrolní úkoly otázky, popř. úlohy k zamýšlení,
- korespondenční úkoly (jen u některých lekcí).

Čas potřebný k prostudování jednotlivých lekcí sice uvádíme, avšak rychlosť studia značně záleží na Vašich schopnostech a studijních návycích. Součástí Vašich studijních povinností je splnění alespoň jednoho korespondenčního úkolu podle pokynů uvedených v tomto textu.

Věříme, že Vám předkládaný studijní materiál pomůže pochopit základní principy modelování a počítačové simulace, a přejeme Vám hodně úspěchů ve studiu.

Autoři děkují prof. Ivanu Křivému a prof. Evženovi Kindlerovi za vytvoření velmi kvalitních skript Simulace a modelování 1 a 2, která byla s jejich laskavým svolením vzata jako základ této inovované opory.

2 Základní pojmy

V této kapitole se dozvíte:

- Základní pojmy
- Systém
- Model
- Modelování
- Simulace

Po jejím prostudování byste měli být schopni:

- Získat povědomí o základních pojmech
- Znát rozdíl v interpretaci modelu
- Vědět, jak se systémy klasifikují a jaké jsou jejich důležité vlastnosti.
- Znát rozdíl mezi modelovaným a modelujícím systémem
- Vědět, co je třeba si zvláště uvědomit při simulaci a modelování.
- Vědět, co je to simulace a modelování v oblasti aplikace výpočetní techniky.

Klíčová slova této kapitoly:

Základní pojmy, modelování, simulace, systém, model.

Doba potřebná ke studiu: 4 hodiny

Průvodce studiem

Jelikož modelování a simulace jsou techniky k exaktnímu výzkumu věcí, které fyzicky existují nebo alespoň mohou existovat, musíme si nejprve vyjasnit základní vztahy mezi exaktními pojmy a jejich obrazy či vzory ve fyzické realitě. K tomu slouží tato kapitola.



*Chceme-li definovat, co znamená modelování a simulace, musíme před tím definovat význam některých výchozích termínů jako **systém** a **model** i termínů pomocných, které pomohou vyjasnit některé nepříliš přesné, avšak všeobecně rozšířené představy. Použité termíny jsou většinou cizí slova, takže je používají i jiné jazyky, zejména mateřský jazyk počítačové profese – angličtina. Je záhadno být ve shodě s tímto jazykem, protože český počítačový odborník je a bude během své budoucí práce nuten číst řadu anglických knih a článků. Současně však je třeba vzít v úvahu, že termíny používané v simulaci a modelování mohou mít jiný význam v běžném jazyku, resp. v odborných terminologích jiných profesí, a na nejdůležitější odlišnosti upozornit.*

2.1 Systém

Slovo systém je v dnešní době používáno v mnoha oborech a v mnoha významech. (Všimněme si např. významové vzdálenosti mezi výrazy operační systém a jazykový systém arabštiny.) Neodpovědným používáním v politice a masmédiích (systémové pojetí, systémové změny, systémový přístup, ...) ztratil tento termín v běžném jazyku téměř všechn význam, stal se prázdnou frází. Jeden obor, který často používá simulaci, totiž teorie





Věc

regulace a technického řízení, vymezuje termín systém dosti přesně (jako objekt se vstupními a výstupními signály svázanými přes své vnitřní stavy pomocí obyčejných diferenciálních nebo diferenčních rovnic), avšak tento fakt nás nesmí svést k tomu, že bychom v simulaci a v modelování chápali systém podobně. Stejně se nesmíme nechat zmást termínem systém, jak už byl dávno zaveden v termodynamice. V simulaci a v modelování jde o něco zcela jiného, jak dále vysvětlíme.

V simulaci a modelování se studuje nějaká **věc**, resp. možné varianty nějaké věci, při čemž slovo věc chápeme tak, jak jej chápou filozofové: je to nějaký objekt hmotného světa, a to buď objekt, který vskutku existuje (např. organismus konkrétní osoby, konkrétní továrna, krajina, škola atd.), nebo o kterém uvažujeme, že by existovat mohl (např. stroj, budova či výrobní provoz, který by měl být realizován, nebo nemocný organismus dané osoby, o jehož terapii se uvažuje, ale definitivní rozhodnutí dosud nebylo formulováno). Věc chápou filosofové v její úplné složitosti (pokud existuje) nebo spolu se všemi nejasnostmi její existence (pokud se uvažuje o možnosti věc realizovat) a chápou i to, že není v lidských silách do detailů celou věc racionálně, tj. rozumovými prostředky, pochopit a zvládnout. Tak to chápou i různé obory vědy, techniky a řízení společnosti, a proto zavádějí na zkoumaných věcech abstrakce, které zanedbávají některé aspekty těchto věcí; zanedbané aspekty jsou vybrány tak, že aspekty, které zbývají, jsou daným vědeckým, technickým či společenským oborem zvládnutelné: mimo jiné, mohou o nich racionálně komunikovat pracovníci odpovídající vědecké, technické či společenské profese.

Systém

Takovou abstrakci budeme v modelování a simulaci nazývat **systémem** a podle charakteru profese, která systém na věci vidí, zavádí či definuje, dostává systém i přívlastek. Systém jako abstrakce nám také umožní se na danou věc dívat s určitou rozlišovací schopností a určit, jak detailně chceme na danou věc nahlížet. Např. zkoumáme-li problematiku dopravního provozu ve městě, nepotřebujeme na jednotlivá vozidla nahlížet na mikroskopické úrovni. Tímto rozlišujeme také určitý stupeň abstrakce.

Příklad:



Např. televizní přijímač neboť i jeho běžný majitel ví, že si ho kupuje pro jeho elektronické je obvykle chápán jako *elektronický* systém, vlastnosti (tedy pro vlastnosti, které patří do profese elektroniky), avšak bytový architekt ho tak chápá nemusí stejně jako např. notář sepisující dědictví po majiteli bytu; nebo železniční síť se běžně chápe jako *dopravní* systém, i když ekolog v ní může vidět systém jiný stejně jako – někdy v minulém století – stavbyvedoucí jejích složek. Na jedné věci lze tedy vidět více systémů.

Úkol k zamýšlení:



Zkuste ve své paměti najít nějaké jiné příklady toho, kdy na "jedné věci" mohou různí lidé „vidět“ několik odlišných systémů. Zkuste např. přemýšlet o nějaké věci z vašeho denního života a uvažujte, jak na ni pohlížejí různé profese a čím se jejich náhledy liší. Tyto profese nemusí být vědecké disciplíny, ale třeba jen různá povolání.

Abstrakce může nebo nemusí zanedbat význam času. Např. význam času v systémech železniční dopravy nelze běžně zanedbat, avšak konstruktér mapy železniční sítě České republiky k jízdnímu řádu roku 2014 zanedbává jak to, že se po jednotlivých tratích pohybují v čase vlaky, tak to, že železniční síť může měnit před rokem, kdy mapu realizuje, i po něm. Systém, v němž se od významu času abstrahuje, se nazývá **statickým systémem** (anglicky **static system**). Pokud se od významu času neabstrahuje, pak jen výjimečně se berou v úvahu i jeho vlastnosti, jak je poznává moderní fyzika. V drtivé většině oborů se čas chápe "newtonovsky", to jest jako v klasické fyzice, čili tak, že je smysluplné mluvit o tom, že dvě události nastaly v systému současně nebo jedna z nich nastala dříve než druhá. Systém, jehož čas se nezanedbává a je přitom chápán takto (newtonovsky), se v modelování a simulaci nazývá **dynamickým systémem** (angl. **dynamic system**). Jak uvidíme později, simulace se jinými než dynamickými systémy nezabývá (Kindler 1980).

Statický systém

Množina okamžiků, v nichž dynamický systém existuje, se nazývá **časovou existencí** tohoto systému; protože v praxi nemá význam mluvit o jiných druzích existence a termín časová existence (dynamického) systému je dlouhý, mluví se krátce o **existenci (dynamického) systému**.

Dynamický

Existence systému



Příklad:

Existence dynamického systému je dána také abstrakcí: např. počítač při nějaké akci (např. při řízení výrobního systému, nebo během doby, kdy na něm píšeme nějaký text, anebo prostě během pracovní doby, tj. od okamžiku, kdy ho při příchodu do práce zapneme, do okamžiku, kdy ho před odchodem z práce vypneme) můžeme chápat jako systém existující jen během této akce, přestože jakožto věc existuje jistě před ní a pravděpodobně i po ní. Množina takových okamžiků nemusí být ani interval reálných čísel: např. pro mnohé odborníky zaměřené na logické obvody jsou zajímavé jen časové okamžiky reprezentující hodinový puls daného počítače a přechodové fáze mezi nimi je nezajímají, nebo pro pracovníka v makroekonomice mohou být důležitá jen data na koncích periodicky se opakujících účtovacích období a od toho, co se děje během těchto období, abstrahuje; pro oba odborníky systém existuje jen v konečné množině navzájem izolovaných časových okamžiků. Tak, jak se to dělá už dávno ve fyzice, lze časovým okamžikům jednoznačně přiřadit polohu na časové ose pomocí reálných čísel. Existence dynamického systému může být v principu jakákoliv neprázdná množina reálných čísel. V praxi jde vždy o množinu dostatečně velikou, což je ovšem mlhavý, ale srozumitelný pojem.

Stav systému
Událost

Dynamický systém je v každém okamžiku své existence v jistém **stavu** (angl. **state**). To, pro co jsme výše použili slova **událost**, je změna stavu dynamického systému. Poněkud nadneseně lze říci, že statický systém je stále v tomtéž stavu; nadnesené je to proto, že se z takového tvrzení nedá nic důležitého odvodit. Moderní fyzika nás učí, že nezanedbáváme-li v systému její poznatky o čase, nemá smysl mluvit ani o jeho stavu v daném čase ani o jeho existenci a události jsou neurčité. Jak už jsme však uvedli, právě poznatky moderní fyziky o čase se v modelování a simulaci neuplatňují.

Prvky systému

V modelování a simulaci se chápe systém tak, že je složen z **prvků** (angl. **elements**). Mezním případem je, že systém má jediný prvek; tato praxe je

však v simulaci poměrně vzácná (konkrétně vzato jen v některých případech, kdy se má simulovat systém definovaný odborníkem v regulaci). Běžně se systém rozkládá na více prvků: známe-li jejich chování, můžeme snadněji porozumět tomu, co se děje v celém systému. Prvky systému, tedy prvky abstrakce na nějaké věci, mohou odpovídat komponentám, které na věci nějak poznáváme fyzicky (např. její jisté prostorové složky – to je v praxi velmi častý případ), logicky (např. schopnosti dané věci či jejích složek), ale simulace neklade žádná omezení na způsob, jak rozklad provedeme; někdy se např. výhodně uplatní, když mezi prvky systému zahrneme i dvojice nebo seznamy jiných prvků téhož systému.

Úkol k zamýšlení:



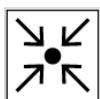
Transakce

Zamyslete se nad případem dynamického systému, který je definován na tanečním parketu, a zkuste si uvědomit, které prvky v něm mohou být. Upozorňujeme, že takový systém lze chápat tak, že obsahuje jednotlivé tanečníky a při tom i jejich páry. Přemýšlejte: oč bychom se ochudili, kdybychom takové dvojice v onom systému zanedbali, a do jakých potíží bychom se dostali, kdybychom v tomto dynamickém systému připustili jen dvojice tanečníků? Jak bychom museli formulovat vystřídání partnerů?

Okolí systému

V dynamickém systému se může počet prvků během jeho existence měnit: systém (např. biologický) může růst a smršťovat se, avšak v technických a ekonomických aplikacích jde nejčastěji o to, že prvky mohou do systému **vstupovat** a systém **opouštět**. Takové prvky se nazývají **transakcemi** (angl. *transactions* nebo **temporary elements**).

Ve skutečnosti takové prvky nevznikají, nýbrž přicházejí do systému z jeho **okolí**, a nezanikají, nýbrž systém opouštějí; avšak vzhledem k tomu, že systém je abstrakce, která našemu rozumu nahrazuje zkoumanou věc, abstrahujeme i od okolí, které sice pro danou věc existuje, ale pro systém nikoliv. Jinými slovy, když je nějaká složka reality přítomna v prostředí, od kterého abstrahujeme, je to v naší abstrakci stejně, jako by neexistovala.



Jako příklady transakcí lze uvést zákazníky vstupující do obchodního domu, pacienty přicházející do nemocnice, zakázky přicházející do výrobního podniku nebo vozidla vstupující do dopravního systému, který je – jakožto systém – abstrahován tak, že je vydelen ze svého okolí, z něhož do něj vozidla přijíždějí a kam jej vozidla opouštějí.



Permanentní prvky
Aktivity

Zkuste najít jiné příklady transakcí. A najdete i příklady transakcí, které opravdu v dané věci vznikají a nepřicházejí do ní z jejího okolí?

Prvky, které jsou v dynamickém systému během celé jeho existence, se nazývají **aktivitami** nebo **permanentními prvky** (angl. **activities** nebo **permanent elements**).

Právě jsme zdůraznili, že když o systému uvažujeme, zanedbáváme vše, co do něho nezahrnujeme. Z toho plyne, že jestliže transakce dynamický systém jednou opustí, je ztracena z našeho uvažování a nemá smysl mluvit o jejím návratu; jinými slovy, pokud bychom připustili, že transakce systém skutečně opustila a pak se vrátila, nemělo by být možno identifikovat, že jde o tutéž transakci. Pokud se identita transakce identifikovat má, pak tato

transakce nemůže systém opustit, nýbrž v něm zůstává, snad nějak skryta a bez důležitosti na dění v systému, avšak nemůže opustit systém, musí být stále do naší abstrakce zahrnuta.

Prvky systému mají své vlastnosti, které se odborně nazývají **atributy**. Příkladem může být *teplota ingotu* v systému oceláren (jde o tzv. **numerický** nebo **reálný atribut**, protože nabývá numerických hodnot, reálných čísel), *funkčnost stroje* ve výrobním systému (jde o tzv. **booleovský atribut**, protože nabývá booleovských hodnot *ano* a *ne*, konkrétněji řečeno *schopen pracovat* a *v poruše*), nebo *jméno zákazníka banky* (jde o tzv. **textový atribut**, neboť nabývá textových hodnot). Atributy tedy přiřazují prvkům nějaké hodnoty a ty se u prvků dynamického systému mohou v čase měnit.

Na první pohled je patrno, že stav dynamického systému v čase t by měl být dán prvky, které jsou v čase t v tomto systému přítomny, a hodnotami jejich atributů v tomto čase. Při bližším rozboru se však ukáže, že stav dynamického systému je ovlivněn i relacemi mezi jeho prvky: je-li například daná zakázka ve výrobním systému zpracovávána na jeho jistém stroji, chápeme přirozeně takový stav výrobního systému za jiný než stav, v němž existuje třeba jen jediná odlišnost, a to ta, že je tatáž zakázka zpracovávána na jiném stroji; odlišná relace mezi zakázkou a strojem má na odlišnost stavů podstatný vliv. V praxi simulace a dalších oblastí modelování se však relace v tom smyslu, jak se jim rozumí v matematice a v operačním výzkumu a jak jsme je právě na příkladu naznačili, nezavádějí. Nahrazují se **referenčními atributy** (angl. **pointers**), totiž atributy, které přiřazují prvkům systému jiné prvky. Např. vztah *zakázka M je právě zpracovávána na stroji P* se reprezentuje jako *hodnota referenčního atributu „zpracovávaná zakázka“ prvku P je rovna M* nebo *hodnota referenčního atributu „zpracovávající stroj“ prvku M je rovna P*. Atributy, které nejsou referenční, se nazývají **standardní**, protože přiřazují prvkům „standardní“ hodnoty (reálná čísla, booleovské hodnoty, texty), přítomné v mnoha systémech, zatím co referenční atributy přiřazují prvkům jiné prvky téhož systému nebo výjimečně „nic“ (obvykle zapisováno jako *none* nebo *nil*).

Kdy je hodnota výše zmíněného atributu *zpracovávaná zakázka* rovna onomu *nic*? Zkuste interpretovat referenční atributy na příkladu systému "taneční parket", se kterým jsme se setkali výše. Konkrétně, místo dvojic zaveděte nějaký referenční atribut, např. *partner*, pro některé osoby. Musí tento atribut mít všechny osoby? Jak to nejlépe vyřešit? Je jasné, že na jedné taneční skutečnosti můžeme „vidět“ dva navzájem odlišné systémy, a to ten obsahující dvojice a pak ten s referenčním atributem *partner*?

Změna hodnoty referenčního atributu znamená změnu konfigurace (struktury) dynamického systému. (V návaznosti na obecné chápání slov „struktura“, resp. „konfigurace“ systému lze tato slova i v oboru simulace chápát jako souhrn všech referenčních atributů prvků systému.)

Kontrolní otázka:

Jaký je rozdíl mezi věcí a systémem? Jak se v systémech vyjadřuje jejich struktura? Může prvek opustit systém a vrátit se do něho? Proč je tomu tak? Jak byste chápali psací stůl, tramvaj, autobus a sklenici v restauraci jakožto prvky jednoho systému? Jak byste odlišili tramvaj od městského autobusu, pokud obojí budete chápát jako systémy pohybující se po ulicích?

Atributy prvků:

numerické
(reálné),
booleovské,
textové

Atributy:
referenční

Atributy
standardní



Struktura systému



2.2 Model

Slova „model“ se používalo v běžné řeči nejprve pro předlohu. V odborném jazyku doby před simulací a virtuální realitou zůstal z této praxe termín „funkční model“, a to pro první exemplář navrženého výrobku, který pracuje tak, jak by výrobek pracovat měl, přestože jiné vlastnosti výrobku (např. estetické) tento exemplář ještě nemá. Z této praxe vznikla i interpretace slova „model“ pro něco zvláštního, nezvyklého či nákladného (např. hlavně před druhou světovou válkou používané termíny „model klobouku“, „model automobilu“ apod.).



Model



Statický model

Simulační model

V modelování a simulaci je termín **model** použit pro analogii (resp. vztah) mezi dvěma systémy. Jednoduché příklady nabízí mapa (model části země na papíře), socha (model osoby, zvířete atd. v neživém materiálu) nebo dětský vláček (model skutečného vlaku ve zmenšeném měřítku). Vztah obou systémů – **modelovaného** a **modelujícího** – je dán tím, že každému prvku P modelovaného systému je přiřazen prvek Q modelujícího systému, každému atributu g prvku P je přiřazen atribut h prvku Q a pro hodnoty atributů g a h je dána nějaká relace. Její charakter není nějak obecně omezen, ale v případě, že g i h jsou aritmetické atributy, bývá taková relace úměrnost, tolerance (mapa zobrazuje zmenšeně a jen přibližně), kombinace úměrnosti a tolerance (např. rozměry složek a částí dětského vláčku jsou přibližně úměrné odpovídajícím rozměrům skutečného vlaku) apod.

Jsou-li modelovaný i modelující systém statické, říkáme, že daný model je **statický model**. V simulaci se však uplatní jen tzv. **simulační modely** (dynamické modely), totiž modely, které splňují následující požadavky:

Jejich modelující i modelované systémy jsou dynamické systémy.

Existuje zobrazení τ existence modelovaného systému do existence modelujícího systému; je-li tedy t_1 okamžik, v němž existuje modelovaný systém M_1 , je mu přiřazen okamžik $\tau(t_1) = t_2$, v němž existuje modelující systém M_2 , a tak je zobrazením τ přiřazen i stavu $S_1(t_1) = \sigma_1$ systému M_1 stav $S_2(t_2) = \sigma_2$ systému M_2 .

Mezi stavy σ_1 a σ_2 jsou splněny požadavky na vztahy mezi prvky a jejich atributy, jak jsme je výše popsali pro modely obecně; jako kdyby každému stavu σ_1 modelovaného systému odpovídal stav σ_2 modelujícího systému tak, že oba stavy jsou ve vztahu statického modelu.

Zobrazení τ je neklesající; pokud nastane stav s modelovaného systému před stavem s^* téhož systému, pak stav, který odpovídá v modelujícím systému stavu s , nastane před stavem, který odpovídá stavu s^* , nebo mohou oba stavy nastat v modelujícím systému současně (totiž v případě, že modelující systém není „tak kvalitní“, aby dokázal zobrazit všechny detaily v modelovaném systému), nikdy však nemůže být časové pořadí stavů v modelovaném systému a jím odpovídajících stavů v modelujícím systému přehozeno.

Čtvrtý požadavek umožnuje tomu, kdo konstruuje modelující systém, aby se při tom nechal inspirovat vztahy kausality v modelovaném systému. Jestliže platí, že nějaké vlastnosti modelovaného systému implikují, že později nastane v tomto systému něco, co ovlivní jeho stav, lze tuto zákonitost napodobit i v modelujícím systému. Příkladem na takový kauzální

(příčinný) vliv může být implikace, že když nějaký permanentní prvek je schopen obsloužit jen jednu transakci a žádají ho o obsluhu dvě transakce brzy po sobě, pak druhá z nich musí čekat ve frontě. Jiným příkladem je to, že když se hodnota nějakého aritmetického atributu mění v čase spojité a je větší než jisté číslo, pak bude větší než toto číslo i v jistém následujícím časovém intervalu.

Model je tedy složitá struktura, která spojuje dva systémy, jejich prvky a jejich atributy, a v případě simulačních modelů i existence obou systémů. Vztah mezi těmito dvěma systémy je tedy odlišný od vztahu mezi věcí a systémem, který na ní „vidíme“. Říkat tedy, že tvoříme model věci tím, když na ní definujeme nějaký systém, je nesolidní. Resp. říkat, že systém je model věci, na které je definován. Později poznáme, že v případě simulace musí být modelující systém (např. proces) definován na věci (např. počítači), která reálně existuje a s níž lze dokonce experimentovat.



Také je nutno pro simulaci zdůraznit, že modelující systém je definován na nějaké věci - odliší se tím simulace od "modelů" chápáných jen jako popisy např. pomocí diferenciálních rovnic, blokových schémat nebo Markovových řetězců - s takovými "věcmi" se těžko experimentuje, i když je např. matematici považují ve své terminologii za modely.

Příklad:



Modelovaný systém je doprava mezi poli a sklady zemědělského podniku, je definován na "věci", kterou je zemědělský podnik se vším všudy tak, jak existuje. Modelující systém je výpočetní proces na počítači, čili je definován na "věci", kterou je počítač tak, jak existuje, např. s fungujícím operačním systémem, který řídí ten výpočetní proces ale i případně jiné procesy, s transportem elektrické energie.

Věc, na které je definován modelovaný systém: zemědělský podnik

Modelovaný (simulovaný) systém: doprava mezi poli a sklady zemědělského podniku

Věc, na které je definován modelující systém: počítač

Modelující (simulující) systém: výpočetní proces na počítači

Model: simulátor dopravy mezi poli a sklady zemědělského podniku

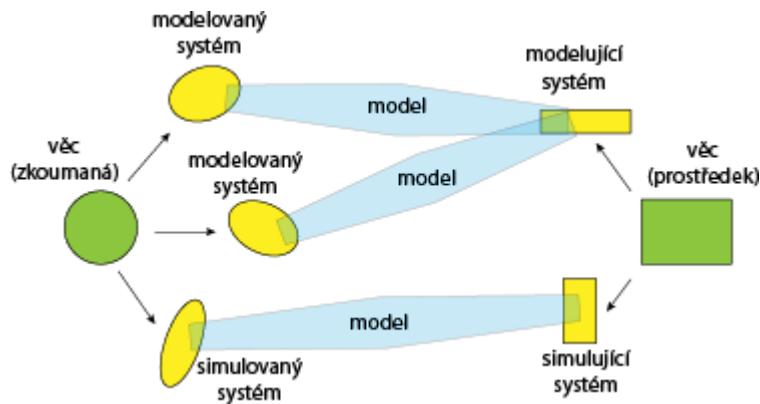
V běžné mluvě se však ustálila praxe, že pod slovem model se rozumí modelující systém. Tato praxe není úplně výstižná a přesná, protože nevystihuje, že model není jen systém, nýbrž že je obrazem „něčeho“ a že to „něco“ zobrazuje „nějakým způsobem“. Je ovšem pravda, že ve většině případů nepřesnost nevadí, a tak se ve světové odborné literatuře (a v některých dalších kapitolách této učebnice) s termínem model na místě modelujícího systému můžeme setkat, aniž by došlo k potížím. Příkladem může být použití slova model při výkladu, jak je modelující systém programován na počítači: místo opakování používání slov modelující systém se mluví prostě o modelu (viz dále). Místo termínu „modelovaný systém“ se používá slova **originál**.



Originál,
Systém
simulovaný
a
simulující.

Stojí za to zmínit, že můžeme provádět modelování bez simulace a simulaci bez modelování. Můžeme tedy vytvořit sochu jako model, s níž neděláme

simulace. Zároveň můžeme provádět simulaci dopravních nehod s reálnými vozidly, aniž bychom vytvářeli model. My se budeme zabývat oběma činnostmi, tedy modelováním a simulací.



Obrázek 2 - Vztah mezi věcí, systémem a modelem

Na (Obrázek 2) můžeme vidět vztah mezi věcí, systémem a modelem. Na začátku je zkoumaná věc reálného světa, anebo věc, kterou teprve hodláme vytvořit. Na tuto věc nahlížíme jako na abstrakci – systém. Pokud nebude v modelu úvahu čas, nazýváme tuto abstrakci modelovaný systém. Pokud faktor času uvažujeme, je tato abstrakce nazývána simulovaný systém. Pokud chceme s modelovaným/simulovaným systémem experimentovat, vytvoříme si jeho model. Model vytváříme na nějaké jiné věci. Tato věc slouží jako prostředek, na kterém budeme experimentovat. Touto věcí může být třeba plastelína (v případě, že nebereme v úvahu čas), anebo např. počítač (kde jsme schopni pracovat i s časem). I s touto věcí jsme schopni pracovat jen prostřednictvím abstrakcí – systému. Proto na této věci definujeme také systém, abstrakci této věci. Pozor! Je potřeba rozlišovat mezi systémem zkoumané věci a systémem věci, kterou používáme jako prostředek. Zatímco v prvním případě půjde o abstrakci míče, na kterém si zavedeme systém koule. V druhém případě půjde o abstrakci počítače, na kterém si zavedeme systém výpočetní proces. Model je potom spojením obou abstrakcí – systémů. Tzn., že model musíme definovat jako vztah mezi modelovaným a modelujícím, případně mezi simulovaným a simulujícím systémem. Vždy musíme říct, jak budeme reprezentovat prvky a atributy jednoho systému do systému druhého.

Model jako takový může být nejen počítačový, ale např. také slovní, myšlenkový, matematický, aj.

Simulátor

V případě, že jde o simulační model, mluví se raději o systému **simulovalém** a **simulujícím** než o modelovaném a modelujícím. Analogicky k právě zmíněné nepřesnosti terminologie existuje praxe, že se na místě termínu simulující systém používá termín **simulační model** nebo také **simulátor**.

Termín *simulátor* nezavádí nepřesnost, a tak ho budeme také v této kapitole používat, přestože ho někteří američtí autoři používají v poněkud užším smyslu (např. jako trenážér). I termín *model* budeme používat ve smyslu simulující systém, pokud bude zřejmé, že tomu tak je.

Kontrolní otázky a úkoly:

Výše jsme v příkladech na statické modely použili rčení, že mapa je model části země na papíře, že socha je model osoby, zvířete atd. v neživém materiálu, a že dětský vláček je model skutečného vlaku ve zmenšeném měřítku. Tak se mluví a bylo by nevhodné chtít na turistech, umělcích, milovnících umění a na hrajících si dětech, aby používali přesné odborné terminologie, kterou musíme respektovat my, (budoucí) profesionálové v exaktních oborech. Jak byste však uvedené příklady vyjádřili přesně v terminologii modelování? Proč nelze pro přesnou terminologii připustit definici, podle které by model byl systém (který cosi modeluje, reflektuje, zobrazuje,...)? A použijme stejného zdůvodnění a zeptejme se: Je videozáznam model? Jak je to správně? O jaký model jde? Když pustíme videozáznam zpětně, o jaký model půjde?



2.3 Modelování

Angličtina je dosti odlišný jazyk od češtiny. Jedna její vlastnost, kterou těžko pochopí ti, jejichž mateřtinou je čeština, spočívá v tom, že druh slova není dán jeho tvarem, nýbrž jeho kontextem. Zatím co v češtině je slovo „model“ podstatné jméno a slovo „modelovat“ sloveso, může být v angličtině slovo *model* podstatným jménem, přídavným jménem i slovesem, a to podle toho, jaká slova jsou před ním a za ním. A stane-li se slovesem, může to být libovolné sloveso, odvozené z podstatného jména model: např. být modelem (tj. např. mapa modeluje Britské ostrovy), ale i např. vytvářet model (přesněji: vytvářet modelující systém, např. sochař Michelangelo modeloval Davida) či používat model, a obojí z různých důvodů; model (resp. modelující systém) může někdo vytvářet prostě jen proto, že má z takové práce potěšení, nebo proto, že se na jeho samotné konstrukci něčemu přiučí, nebo proto, aby ho později k něčemu použil; a pod slovem „použít“ se může skrývat např. použít k zjištění něčeho o originálu, použít k vlastnímu potěšení, použít k trénování práce s originálem, použít jako náhradu originálu (jakousi protézu) v běžném životě atd. A to vše (a mnoho jiného) lze v angličtině shrnout pod sloveso *to model* (tedy *modelovat*).



Příklady:

V češtině dostalo slovo *modelovat* postupem doby několik významů. Jeden z nich je „dávat věci nový prostorový tvar“; např. geografové říkají, že řeka svou erosivní činností modeluje krajinu, nebo historik umění použil výrazu „malování bylo doplněno modelováním“, když chtěl říci, že v jisté oblasti se nejprve porcelánové nádoby zdobily malbou květinových motivů, ale později byly malované květiny naznačeny na povrchu i plasticky; i děti používají někdy slova modelovat v podobném smyslu o své zábavě.



Slova model a modelovat byla v posledních desetiletích používána tak často a tak nezodpovědně, že ztratila téměř všechn význam. Tento proces proběhl i v řadě vědeckých oblastí, a tak dnes nemá metodologie vědy ve věci termínu „modelování“ vůbec jasno. Neexistuje všeobecně přijatá definice a mezinárodní odborné akce spíše „mapují“, co vše se pod tímto termínem rozumí: ekologové, ekonomové, sociologové, chemici, astrofyzici, kosmologové, jaderní fyzikové, fyzikové pevné fáze, statistikové, logikové

Modelování jako výzkumná technika



a další obory s námahou zjišťují, v čem se při porozumění termínům „model“ a „modelování“ shodují a v čem se různí.

V oblasti, které se dříve říkalo kybernetika a která se dnes zaplňuje systematickými aplikacemi výpočetní techniky, dominuje anglicky psaná literatura, které je nutno se přizpůsobit, a tedy si uvědomit, že modelovat (anglicky *to model*) a „modelování“ (angl. *modelling*, případně – od amerických autorů – *modeling*) může mít takřka neomezeně mnoho významů, jak jsme výše naznačili. Můžeme však ještě dodat, že tehdy, kdy jde o modelování ve smyslu **výzkumné techniky** (nebo – jak se někdy říká – **metody poznání**), je už obsah tohoto termínu vymezen jasněji, a to v následujícím smyslu (Kolektiv autorů 1986):

Podstatou modelování ve smyslu výzkumné techniky je náhrada zkoumaného systému jeho modelem (přesněji: systémem, který jej modeluje), jejímž cílem je získat pomocí pokusů s modelem informaci o původním zkoumaném systému.

V tomto smyslu tedy platí, že vytvoříme model, v němž modelovaným systémem je zkoumaný systém, ale my budeme experimentovat s modelujícím systémem, při čemž cílem bude dozvědět se něco o modelovaném systému. Pokud by cílem bylo pouhé vytvoření modelu, resp. modelujícího systému, šlo by o modelování, ale jakožto zábavu a ne ve smyslu výzkumné techniky. Pokud by cílem bylo nahrazení modelovaného systému modelujícím systémem v reálném životě, šlo by o modelování ve smyslu vytváření protézy, a pokud by cílem experimentování bylo dozvědět se něco o modelujícím systému bez vztahu k systému modelovanému, vypadl by model úplně „ze hry“ a šlo by vlastně jen o přímé experimentování s modelujícím systémem (příkladem může být, když si děti na hračkách nebo na počítačové obrazovce modelují dejme tomu srážky vlaků nebo jejich střety s vesmírnými nestvůrami).

Modelování jakožto široký obor aplikací výpočetní techniky je takřka výhradně zaměřeno na to, co jsme výše akcentovali: na modelování ve smyslu výzkumné techniky. Je však vhodné uvědomit si bytostný fakt, že modelování v tomto smyslu není na aplikace výpočetní techniky omezeno. Modelující systém může být abstraktní matematická struktura (vzorec, ...) manipulovaná lidskou myslí a interpretovaná třeba na papíře, může to být fyzikální analogie (např. hydrodynamická analogie elektrického procesu nebo tzv. Bohrův model atomu) apod. Je ovšem pravda, že v dnešní době se z mnoha důvodů stále více uplatňuje ve funkci modelujícího systému výpočet na číslicovém počítači. Při tom je vhodné si uvědomit, že zde platí jistá analogie s automatizací opravdových pokusů (pokusů se zkoumaným objektem a ne se systémem, který ho modeluje): podobně jako moderní experimentátor nemusí zkoumaným objektem osobně manipulovat, ale může řadu pokusů automatizovat, jmenovitě pomocí řídícího počítače, který pokus či posloupnost pokusů řídí a vyhodnocuje, tak ani pokus s počítačově realizovaným modelujícím systémem nemusí být interaktivním dialogem mezi počítačem a operátorem, nýbrž může být naprogramován jako série automaticky řízených, za sebou následujících pokusů; na tomtéž počítači, kde je realizován modelující systém, může být realizována i automatická manipulace s tímto modelujícím systémem, obojí dokonce může být integ-

rováno do jednoho výpočtu. Blíže se k celé věci (včetně příkladů) vrátíme, až budeme probírat, co je simulace.

Kontrolní otázky:

Pokuste se najít nějaký výpočet na počítači, který není modelujícím systémem v žádném modelu. A vzpomeňte si, zda jste vy nebo vaši kamarádi někdy modelovali bez použití počítače. Co vše byste mohli říci ke slovnímu výrazu *statistický model* v případě, že jde např. o určení vlastností dat získaných při opakovaných měřeních téhož jevu? Je hledání nejkratší cesty mezi dvěma místy pomocí mapy modelováním? Můžete něco podobného prohlásit o práci s počítačem, který vám má sdělit nejkratší vlakové spojení mezi dvěma místy, začínající v daném okamžiku nebo později?



Na závěr ještě poznamenejme, že odborníci, kteří mají blízko k modelování, často používají slova modelovat ve smyslu *být modelujícím systémem* něčeho. Řeknou např., že křivka modeluje průběh epidemie nebo že mapa modeluje část zemského povrchu. Jde o odbornou „hantýrku“, která mezi školenými odborníky nemusí zavádět nejasnosti, avšak raději se jí vystříhejme.



2.4 Simulace

V obecné mluvě značí simulace předstírání nemoci, bezvědomí, duševní poruchy apod. Profesionálně vzato, tento význam slova „simulace“ by měl patřit někam do sociální psychologie. Lze to chápat jako odborný termín, ovšem ne v informatice ani v příbuzných oborech. Pojem „simulace“, jak ho chápe aplikovaná informatika a kybernetika a jak ho chápou i ostatní obory, když aplikují výpočetní techniku, má však zcela jiný obsah. Stručně řečeno, v této oblasti je simulace chápána jako modelování ve smyslu výzkumné techniky, při němž použitý model je simulační. Zrekapitulujme viz. (Kindler 2004, Kolektiv autorů 1986):

Simulace je výzkumná technika, jejíž podstatou je náhrada zkoumaného dynamického systému jeho simulátorem s tím, že se simulátorem se experimentuje s cílem získat informace o původním zkoumaném dynamickém systému.



V tomto smyslu je třeba termín „simulace“ dále chápat. Všimněme si, že zde platí vše, co bylo řečeno výše o modelování jakožto výzkumné technice. Předně cílem simulace je získat informace o simulovaném systému, zatímco pouhá jeho náhrada simulátorem nestačí. Taková náhrada se někdy nazývá **emulací** – na příklad simulátor jednoho počítače P_1 realizovaný na počítači jiného typu je jakousi protézou, která nahrazuje P_1 – ten např. není pro nás dostupný, ale máme pro něj programy. Za druhé simulátor nemusí být realizován na číslicovém počítači, ale dnes je takto realizován stále častěji. Číslicový počítač má výhody v tom, že je dálkově dostupný přes síť, že se dá použít i k jiným účelům (čehož lze využít, když zrovna nemáme důvod použít ho k simulaci), že nekazí životní prostředí a nespotřebuje mnoho energie; nezanedbatelná výhoda číslicového počítače je i v tom, že výpočty na něm (a tedy i pokusy se simulátorem) lze reprodukovat.

Emulace

Zdůrazněme, že aby šlo o simulaci, musí být cílem experimentů se simulátorem snaha dozvědět se něco o simulovaném systému. Když je si-

mulátor realizován jako výpočet na číslicovém počítači, může se složkou simulace stát i experimentování se simulátorem, jehož cílem je získat informaci o něm samotném a ne o simulovaném systému: nastane to tehdy, když např. zjišťujeme, zda v příslušném programu není programátorská chyba nebo zda v něm není použita nevhodná numerická metoda; tento proces se nazývá **ověření správnosti modelu** nebo (krátce) **verifikace modelu**. To samo ovšem simulace není, je to pomocná fáze před simulací, kterou často předchází. Verifikaci modelu následuje ještě tzv. **validace modelu**, tedy **ověření pravdivosti modelu**. Tehdy ověřujeme, zda náš návrh modelu opravdu reflektuje chování „věci“ (resp. reálného systému).

Validace

Validace nám pomáhá zjistit, zda návrh modelu reflektuje chování reálného systému. Jedná se o ověření vztahu mezi realitou a navrženým modelem. Můžeme např. ověřit, že když vypneme pravidlo pro přechod na alternativní zdroje energie, dojde k rychlejšímu vyčerpání ropy.

Verifikace

Verifikace nám poté pomáhá ověřit správnost modelu, programátorské chyby. Jedná se o ověření vztahu mezi abstraktním návrhem modelu a konkrétní implementací. Pokud by šlo např. o pravidlo přechodu na alternativní zdroje, ověříme, zda implementace odpovídá naši představě o fungování daného pravidla.

Zatímco validace je do jisté míry subjektivní, jelikož žádný model nemůže plně odpovídat realitě, lze verifikaci provádět vcelku exaktně. Při validaci hodnotíme tedy zejména užitečnost modelu, hodnotíme jeho strukturu a chování vzhledem k reálnému systému. Při verifikaci zkoumáme již námi vytvořený model, zda se chová tak, jak předpokládáme, že by se na základě námi vytvořené představě o chování reálného systému chovat měl.

Sloveso **simulovat** (angl. **to simulate**) může mít větný předmět, ale nemusí. Např. lze říci, že jistá kniha referuje o tom, jak simulovat (tedy simulovat nemá předmět), nebo že náplní práce jistého zaměstnance je simulovat výrobní procesy (tedy simulovat koho-co). Ve shodě se světovou odbornou literaturou budeme první alternativu (bez předmětu) chápat jako *realizovat simulaci* (ve smyslu právě vymezeném), v druhém případě jako zkoumat daný předmět pomocí simulace (včetně všech jejích složek, tj. i případně včetně vytváření simulátoru, jeho verifikací apod.). Odborník, který simuluje, je v angličtině nazýván **simulationist**, jednoduchý český termín neexistuje. Stejně jako se v odborné hantýrce používá slovo modelovat ve smyslu být modelujícím systémem něčeho, používá se ve stejně hantýrce i slovo simulovat ve smyslu být simulačním modelem něčeho – např. že počítač simuluje námořní bitvu nebo elektronický obvod simuluje nárušt znečištění životního prostředí. Také v tomto případě doporučujeme se k použití hantýrky nepřipojovat.



Některá spojení jako např. *statická simulace*, která se dříve vyskytla tu a tam hlavně v německé literatuře, je nejlépe ignorovat, neboť dnes působí paradoxně, a tedy neodborně (podobně jako např. „hranatá koule“); stejně lze doporučit vyhýbat se opačným výrazům: např. dynamická simulace je stejně užití nadbytečných slov jako hranatá krychle.

V profesi simulace je vhodné chápat rozšíření pojmu simulace k pojmu modelování tak, že u modelování chybí čas, což ovšem může nastat i tak, že pořadí událostí v modelovaném systému není respektováno v pořadí

odpovídajících událostí v modelujícím systému (vzácný ale přesto možný případ). Typickým případem modelování v tomto smyslu, které není simulací, je práce s tradičními CAD systémy, kde s modelujícím systémem různě kroutíme, abyhom ho viděli ze všech stran, necháváme rotovat, počítáme jeho moment setrvačnosti, namáhání atd.

2.5 Simulace na počítačích

Ve starých dobách byl simulátor realizován na speciálních zařízeních a podle nich dostávala příslušná simulace přívlastek: elektromechanická, hydrodynamická, mechanická, odporová, galvanická, analogová (pomocí analogových počítačů) a hybridní (pomocí hybridních analogo-číslicových počítačů). Dnes vytlačila všechny tyto druhy simulace, při níž je simulátor realizován na číslicovém počítači, tedy simulace **číslicová** (angl. **digital simulation**). V americké odborné literatuře se často ve stejném smyslu používá i termín **computer simulation**. Nejde o simulaci počítačů, nýbrž o něco jiného; doslový český překlad by zněl počítačová simulace, což nezní ani jasně ani pěkně – raději bychom řekli **simulace na počítačích**. Ovšem jak už jsme naznačili v oddíle o modelování, počítače mají mnoho výhod proti jiným věcem, na nichž lze realizovat simulátory, a tak je jiná než číslicová simulace už téměř zcela vytlačena za hranice současné vědy a techniky. Také v dalším výkladu půjde jen o tento typ simulace, a tak se nyní s čtenáři domluvíme, že od této doby budeme přívlastek **číslicová** vymýchat.

Nějaké přesnější vyjádření charakteru simulace podle výše zmíněných pravidel pro její přívlastek se dnes nepoužívá: nepíše se o simulaci osobněpočítačové, pentiové, síťové či multiprocesorové. Když je však jasno, že jde o simulaci číslicovou, spojuje se možnost vyjádřit něco přívlastkem s něčím zcela jiným, totiž s charakterem simulovaného (!) systému. Jestliže ten je spojitý, tj. jestliže se hodnoty jeho atributů mění v čase jen spojitě, mluví se o **spojité simulaci** (angl. **continuous simulation** nebo **continuous system simulation**, tj. simulace spojitých systémů). Jestliže je simulovaný systém diskrétní, tj. nenastávají v něm spojité změny v čase, mluví se o **diskrétní simulaci** (v anglické literatuře se používá téměř výhradě termínu **discrete event simulation**, tedy simulace diskrétních událostí). Je-li simulovaný systém tak říkajíc kombinovaný, to jest má-li jak vlastnosti typické pro spojité systémy, tak vlastnosti typické pro diskrétní systémy, mluví se o **kombinované diskrétně-spojité simulaci** nebo častěji prostě o **kombinované simulaci** (angl. **combined simulation**).

Významy právě zavedených tří základních větví (číslicové) simulace jsou v různých situacích a různými autory více či méně modifikovány nebo i komoleny, proto potřebují blíže vysvětlit.

Připomínáme, že systém je definován na věci. Na jedné a též věci může být definován jak spojitý, tak diskrétní (případně i kombinovaný) systém. A tak se nesmíme divit, když je někdy jedna věc zkoumána pomocí spojité, diskrétní a případně i kombinované simulace (např. odborník v oboru elektronických obvodů nebo polovodičové fyziky „vidí“ na počítači spojitý systém, a může tedy např. zkoumat procesor pomocí spojité simulace; avšak odborník v oboru hradlové logiky vidí na počítači diskrétní systém a může



Číslicová simulace

Spojité simulace

Diskrétní simulace

Kombinovaná simulace



aplikovat na studium téhož procesoru diskrétní simulaci. Připomínáme dále, že i když simulujeme spojity systém na číslicovém počítači, nesmí nás zmást fakt, že „uvnitř počítače“ existuje jakýsi diskrétní dynamický systém, vzniklý aplikací numerické metody a – jak se běžně říká – diskretizací modelovaného spojitého systému: v takovém případě jde o spojitu simulaci, protože – jak už jsme uvedli – přívlastek reflektuje to, jak my definujeme simulovaný systém, a ne to, co se děje v simulátoru.

Kontrolní úkol:



Máte alespoň povrchní představu o tom, jak počítač naprogramovat tak, aby se stal simulátorem rakety, boje policie s překupníky drog nebo ocelářského podniku? Zkuste se nad tím zamyslet. Nejde o žádnou fantazii, takové simulátory už dávno existují. Později si o technice jejich tvorby řekneme více.

2.6 Termíny používané při číslicové simulaci

Simulační program

Program, který řídí výpočet při (číslicové) simulaci, se nazývá **simulačním programem**. V jedné věci není ve světě vůbec jednoty, totiž zda se pod tímto termínem rozumí program ve strojovém kódu, který skutečně výpočet řídí, nebo program v programovacím jazyku, jak ho napiše jeho autor. Zdá se však, že důvod této nejednotnosti spočívá v tom, že v praxi kvůli ní nedochází k žádným fatálním nedorozuměním. A tak i v těchto materiálech budeme používat termín *simulační program* jak pro text, který napiše autor simulačního modelu v programovacím jazyku, tak pro strojový kód, který z něho vznikne komplikace čili automatickým převodem do strojového kódu (interpretace zdrojového textu se dnes při simulaci pro svou zdoluhavost téměř nepoužívá). Je vhodné si uvědomit, že simulační program není simulátorem – tím se stane počítač, když je tímto programem řízen!

Simulační pokus

Pokus se simulačním modelem se nazývá **simulační pokus** (angl. **simulation experiment**). V české literatuře se často vyskytuje termín *simulační běh*, ale ten nemá žádnou analogii v literatuře ve světových jazycích. Slovo *run* (tedy běh) se navíc hodí jako protiklad ke slovu komplikace, resp. překlad (např. „chyba zjištěná při překladu“ versus „chyba při běhu“), a – jak dále poznáme – při běhu neběží jen simulační pokusy.

Simulární či
simulovaný čas
versus reálný
čas výpočtu

Když na počítači běží simulační pokus, je záhadno evidovat při něm i čas, který by dané fázi výpočtu odpovídal v simulovaném systému, a to dejme tomu na adrese *time* (což je anglicky *cas*). Když je na této adrese hodnota T , pak výpočet jakoby sděloval „tedy“ by měl být v simulovaném systému čas T . Vzhledem k tomu, že v simulačním modelu nesmí být pořadí odpovídajících si stavů v simulovaném a simulujícím systému přehozeno, nesmí obsah adresy *time* během simulačního pokusu klesnout, nýbrž musí občas povyrůst. Mezinárodní autority v oboru simulace doporučily, aby se tyto hodnoty nazývaly **simulárním časem** (**simular time**), avšak odborná veřejnost tvrdošíjně používá ne zcela přesný, ale všeobecně rozšířený termín **simulovaný čas** (angl. **simulated time**). Je nutné si uvědomit, že simulovaný čas není zdaleka totéž, co **reálný čas**, v němž simulační pokus běží, avšak během takového pokusu se nemůže stát, že by hodnota simulovaného času klesla, zatím co by reálný čas pokročil, nebo naopak.

Mezi dvěma po sobě následujícími simulačními pokusy může ovšem simulovaný čas klesnout (jelikož simulační pokus spouštíme vždy od začátku), přesto že reálný čas celé simulační studie pokračuje dále k vyšším hodnotám.

Posloupnost simulačních pokusů majících stejný účel se nazývá **simulační studie** (angl. **simulation study**). V dnešní době je často realizována jako jeden výpočet (task). Před každým pokusem se simulovaný čas vrátí zpět a rovněž se změní některé hodnoty způsobem, který nemá vzor v simulovaném systému (např. se vyprázdní fronty a vynulují některé součty). Navázání jednoho simulačního pokusu na druhý lze tedy nejlépe chápout jako analogii toho, že simulovaný systém zmizí a místo něho přijde v potaz systém zcela nový, který z historie původního systému „nedědí“ vůbec nic. Jeden „běh“ (ve smyslu použití zkompilovaného programu – viz výše) odpovídá tedy nejčastěji jedné simulační studii, tedy několika simulačním pokusům (kdybychom použili výše zmíněného zlozvyku nazývat simulační pokus simulačním během, museli bychom přijmout bizarní tvrzení, že jeden běh se může skládat z více simulačních běhů).

Ve světové literatuře se zavádí ještě termín **simulační krok** (angl. **simulation step**), a to pro časový úsek výpočtu, během něhož se nemění hodnota simulovaného času. Simulační studie se tedy skládá ze simulačních pokusů a ty se skládají ze simulačních kroků; na začátku každého pokusu se simulovaný čas vrátí na svou výchozí hodnotu (obvykle na hodnotu nula) a – s výjimkou prvního simulačního kroku každého simulačního pokusu – se na počátku každého simulačního kroku zvětší hodnota simulovaného času o nějaký nezáporný přírůstek. Je-li tento přírůstek pro celý simulační pokus stejně velký, mluví se o **ekvidistantním simulovaném čase**, v ostatních případech se mluví o **neekvidistantním simulovaném čase**.

Zvláštním případem je tak zvaná **simulace v reálném čase** (anglicky **real time simulation**). O té se mluví, když simulovaný čas a reálný čas rostou tak podobně, že ten, kdo operuje s počítačem, nezaregistrouje jejich případné malé rozdíly.

Kontrolní otázky:

1. Co je to systém a na čem je definován?
2. Jak je definován model a jaký je rozdíl mezi modelem a simulujícím modelem?
3. Jaký je rozdíl mezi modelováním a simulací?
4. Co je to simulátor?

Simulační studie

Simulační krok

Čas ekvidistantní a
neekvidistantní

Simulace
v reálném čase



Korespondenční úkol:

Zvolte si nějaký více nebo méně exaktní prostředek, např. jakýsi programovací jazyk, blokové schéma apod. A snažte se pomocí tohoto prostředku popsat nějaké systémy nebo alespoň některé jejich prvky. Zkuste takto popsat nějaké statické systémy ze svého okolí a nějaké dynamické systémy. A zkuste vyjádřit více méně přesně i pravidla, podle nichž se prvky v systému chovají a působí na jiné prvky. Tento pokyn platí zejména pro



dynamické systémy, ale pravidla chování lze nalézt i u statických systémů. Když si s nějakou složkou vašeho popisu nebudete umět poradit tak, abyste ji vyjádřili přesně, popište ji v přirozeném jazyku, avšak snažte se co nejvíce porozumět jemnějším detailům takové složky (a případně alespoň tyto složky popište poněkud exaktněji). Uveďte nějaké (hypotetické) příklady odlišného vývoje simulovaného času a reálného času (tj. stanovte si rychlosť počítace a tok událostí v simulovaném systému a porovnejte, tak by asi vámi odhadnuté hodnoty reálného a simulovaného času korespondovaly).



Úkoly k zamýšlení:

1. V jakých souvislostech jste se setkali s termínem model a co vyjadřoval?



Shrnutí obsahu kapitoly

V této úvodní kapitole jste se seznámili se základními pojmy z oblasti modelování a simulace systémů. Jde především o pojmy "systém a jeho okolí", "prvek systému a jeho atributy", "model systému jako vztah mezi dvěma systémy" (modelovaným a modelujícím), "simulační model", "modelování" a "simulace". Je velmi důležité, abyste zavedené pojmy správně pochopili. Věnujte této části mimořádnou pozornost a teprve až se ujistíte, že jste všemu porozuměli, přistupte ke studiu následujících kapitol.

Věc je objekt hmotného světa se vším všudy tak, jak existuje. Systém je vždy definován na nějaké věci a je abstrakcí dané věci. Systém může být statický či dynamický, v závislosti na tom, zda v dané abstrakci uvažujeme s časem. Model je definován jak vztah mezi dvěma systémy. Mezi modelovaným a modelujícím systémem, případně mezi simulovaným a simulujícím (simulačním) systémem. Statický model definujeme např. mezi modelovaným a modelujícím systémem. Pojem model používáme jak pro vztahy mezi statickými i dynamickými systémy, pojem simulátor používáme pro vztah mezi dynamickými systémy.

3 Komplexní systémy

V této kapitole se dozvíte:

- Jednoduché a složité
- Charakteristiky komplexních systémů
- Studium komplexních systémů
- Zpětná vazba
- Simulace

Po jejím prostudování byste měli být schopni:

- vědět, co jsou to komplexní systémy a jak je studovat.

Klíčová slova této kapitoly:

Základní pojmy, modelování, simulace, model.

Doba potřebná ke studiu: 4 hodiny

Průvodce studiem

Náš svět se stává ze dne na den komplexnějším. Stejně tak prostředky, kterými se snažíme popsat a řídit okolní svět, jsou také čím dál tím více složitější. Přesto, ve velkém množství případů uvažujeme stále lineárně a mnohdy jsme zaskočeni chováním okolních systémů, které jsou charakteristické jistou dávkou nonlinearity. Komplexní systémy můžeme najít ve všech oblastech lidské činnosti od filozofických a humanitních věd až k exaktním vědám jako matematika, informatika a fyzika. Modelování a simulace nám poskytuje efektivní nástroj jak vůbec pochopit a vypořádat se s chováním těchto systémů.



3.1 Komplexní systémy

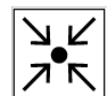
Na každý reálný systém je možno nahlížet jako na **komplexní fenomén**. Je složen z množství vzájemně propojených částí, které spolu reagují takovým způsobem, že může být nesnadné je separovat. Navíc, chování systému je obtížné odvodit z chování jeho částí a toto odvození je tím obtížnější, čím více částí, nebo čím více interakcí se v systému nachází. Vlastnosti systému mohou být rovněž výsledkem interakcí mezi komponentami systému, které tyto vlastnosti nemají. Tento jev nazýváme **emergencí**, kdy vzájemné lokální působení mezi entitami způsobí na globální úrovni úplně nový fenomén. Tento nový fenomén se z dílčích elementárních chování a vztahů vynořuje naprostě nečekaně a většinou i bez náznaků nějakého centrálního řízení.



Komplexní fenomén

Příkladem může být např. tvar a chování hejna ptáků. I přesto, že zde není žádné centrální řízení a mezi jednotlivými jedinci dochází pouze k lokálním interakcím, vykazuje hejno jakoby organizovanou strukturu a chování.

Čím složitější je systém z hlediska komponent a interakcí, tím složitější je jej dekomponovat na jednotlivé části. Navíc, v případě, že chování a vlastnosti systému emergují z interakcí jeho částí, jeví se jako vhodnější použít integrativní přístup namísto redukčního. To znamená, že se na systém díváme z širšího úhlu pohledu (opak specializace). Což není v podstatě nic



jiného, než rozdíl mezi přístupem **shora dolů** (*top-down*) a **zdola nahoru** (*bottom-up*). Dokonce se ukazuje, že navrhovat např. inteligentní systém je výhodné zdola-nahoru, kdy výsledná inteligence emerguje z chování jednotlivých částí, např. reaktivních agentů. Systémem se pak zabýváme zejména z pohledu jeho struktury a chování jako celku, což je opak přístupu shora dolů, kdy se snažíme rozdělit systém na jednotlivé části. Příkladem může být emergence koordinované komunikace v populaci agentů.



Složitost systému C_{sys} (Gershenson 2002) se zvyšuje s množstvím jeho komponent $\#\bar{E}$, interakcí $\#\bar{I}$ mezi komponenty, složitostí samotných elementů C_e a složitostí interakcí C_i (1).

$$C_{sys} \sim \begin{cases} \bar{E} \\ \bar{I} \\ \sum_{j=0}^{\#\bar{E}} C_{e_j} \\ \sum_{k=0}^{\#\bar{I}} C_{i_k} \end{cases} \quad (1)$$

Nelze jednoduše a bez kontextu rozhodnout, kdy je systém ještě jednoduchý a kdy již komplexní.



Například celulární automaty jsou poměrně komplexní systémy svou strukturou, ale s celkem jednoduchými interakcemi. Na druhou stranu, systém pohybu kyvadla je strukturou poměrně jednoduchý systém, ale s poměrně složitými interakcemi (složitým chováním).



V případě statického systému je jednou z vhodných možností využít optimalizace atributů modelu v rámci běhu simulačního pokusu. **Optimalizace** nám dává ty nejlepší výsledky pro danou konfiguraci. Např. při návrhu křídla letadla nabízí optimalizace dobré výsledky, jelikož aerodynamické nároky letadla jsou konstantní (pro danou rychlosť, hmotnosť, atd.). Nicméně, optimalizační techniky mají tendenci dávat zastaralá řešení v případě, že problém se změní dřív, než stačíme provést samotnou optimalizaci.

Komplexní systém



Naproti tomu dynamické **komplexní systémy** se typicky sestávají z množství propojených pozitivních a negativních zpětnovazebních smyček, kde změna jakékoliv jeho části vyvolává kaskádovou změnu v množství propojených komponent, což se v konečném důsledku projeví negativní nebo pozitivní vazbou v původní komponentě. V případě proměnného času mezi těmito efekty je nemožné provádět jakoukoliv optimalizaci takového systému. Jelikož nevíme, co bude ovlivněno jako první, tím pádem ani nevíme, zda efekt bude potlačen či zesílen.



Jako příklad můžeme uvést prostředí burzy. Na burze jsou akcie prodávány či nakupovány v závislosti na jejich ceně. Cena se odvíjí od toho, jak hodně se nakupuje nebo prodává. Tato vnitřní zpětnovazební smyčka má jak pozitivní, tak negativní aspekt. Zákon nabídky a poptávky implikuje negativní zpětnou vazbu, jelikož zvýšení ceny běžně redukuje poptávku, která po určité době, opět sníží cenu. Paralelní mechanismus spekulací přesto způsobuje pozitivní zpětnou vazbu, jelikož rostoucí cena vytváří u nakupujících předpoklad, že v růstu bude pokračovat, čímž je motivuje k dalšímu nakupování. Interakce mezi těmito dvěma nelineárními efekty jsou běžné na trzích a produkují chaotické chování ve vývoji ceny.

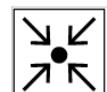
Nikdy tedy nebudeme schopni plně kontrolovat nebo předvídat chování takovýchto komplexních chaotických systémů. Vždy se vyskytnou chyby, překvapení a nečekané problémy. Jaké je tedy jedno z možných řešení? Můžeme se vyrovnat s nečekanými událostmi **adaptováním** našich akcí vzhledem k vzniklé situaci. Abychom se adaptovali na změnu (očekávanou nebo neočekávanou), je dostatečné **kompenzovat** vzniklou **odchylku** aktuální situace od požadovaného směru. V případě, že reakce na změnu přijde dostatečně rychle (předtím než problém bude mít možnost narůst), tato regulace (*feedback-based*) může být velmi efektivní. V případě, že máme prostředky k potlačení vzniklé odchylky chování systému od požadovaného směru, tak již nezáleží, jak moc je samotný systém komplikovaný z pohledu faktorů a interakcí.



Např. nezáleží, jaké kombinace komplikovaných sociálních, politických nebo technologických změn způsobí změnu chování ekonomiky. Centrální banka může regulovat tempo růstu zvyšováním nebo snižováním úrokových sazeb.

Abychom byli schopni reagovat rychle a pružně, je dobré mít alespoň nějaká **očekávání** toho, co se může stát a jaké reakce v řízení systému budou odpovídající. Očekávání zde reprezentují subjektivní pravděpodobnosti, které se učíme ze zkušeností. Potřebujeme tedy mít znalost o systému.

Čím častěji se okolnost B objeví po okolnosti A (anebo čím účinnější je akce B při řešení problému A), tím silnější bude asociace $A \rightarrow B$. Střetneme-li se v budoucnu s A (nebo s něčím podobným), budeme připraveni a pravděpodobně budeme i adekvátně reagovat. Jednoduchým uspořádáním možností dle pravděpodobností, se kterou mohou nastat, tak nesmírně snižujeme složitost při rozhodování.

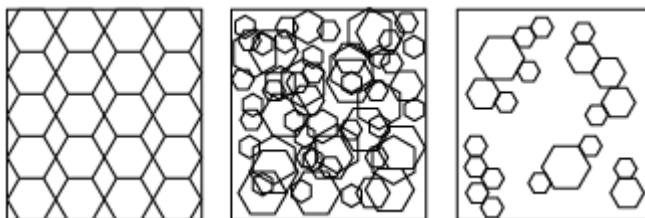


3.2 Jednoduché a složité systémy

Z předchozí kapitoly víme, že komplexní systém se skládá z množství prvků, které mají mezi sebou množství interakcí. Dále záleží na tom, jak moc jsou složité jednotlivé prvky a jejich interakce a zároveň jaký je počet těchto prvků a interakcí. To nám v podstatě rozděluje systémy na jednoduché a složité (Pelánek 2011).



Rozdělení systémů lze jednoduše ilustrovat na příkladu pomocí tzv. komunikačního cvičení (Obrázek 3). Máme dvě osoby. Úkolem první bude popsat situaci, kterou vidí na obrázku. Úkolem druhé osoby, které nevidí obrázek, je dle pokynů první osoby tento obrázek reprodukovat tak, aby byl co nejpodobnější originálu.



Obrázek 3 - Ilustrace tří typů systémů: jednoduchý, neorganizovaný, komplexní

Reprodukovaný první obrázek je vcelku jednoduché. Nacházejí se zde stejně velké trojúhelníky v pravidelné mřížce. Druhý obrázek je chaotický, nevidíme zde žádné pravidelnosti, ani struktury. Přesto je možno tento obrázek reprodukovat nakreslením cca 50 trojúhelníků proměnné velikosti a výsledný obrázek bude velmi podobný originálu. Poslední obrázek tvoří zřejmě struktury, ale jejich popis nebude až tak jednoduchý, jelikož jej nelze popsát pomocí souhrnných informací (Tabulka 1). Tvoří sice pravidelnosti, ale ty nejsou jednoduše popsatelné.

Tabulka 1 - Ilustrace tří typů systémů: jednoduchý, neorganizovaný, komplexní

Systém	Počet prvků	Interakce	Metody řešení problému
jednoduchý	malý	pravidelné	dedukce
neorganizovaný	velmi velký	nahodilé	statistika
komplexní	velký	organizované	simulace, indukce

Jednoduchý systém

Jednoduché systémy mají tedy malý počet prvků a pravidelné interakce. Je pro ně charakteristická pravidelnost jak v uspořádání prvků, tak v chování. Příkladem jednoduchého systému je např. páka, kladkostroj, nebo koloběžka. Tyto systému můžeme analyzovat pomocí dedukce. Vyjdeme z předpokladů a pomocí logických úvah a řešení jednoduchých rovnic dojdeme k jasným závěrům.

Neorganizované systémy jsou tvořeny obrovskými množstvím prvků s nahodilými interakcemi, nelze je popsát úplně přesně, ale mají málo pravidelností. Interakce mezi prvky jsou nahodilé, jednoduché a nepravidelné. Příkladem jsou např. plyny, které se skládají z velmi velkého počtu molekul, u nichž dochází k nahodilým srážkám. Chování systémů approximujeme a pracujeme se sumárními proměnnými, které vyjadřují vlastnosti velkého počtu zprůměrovaných částí. **Komplexní systémy** jsou tvořeny s velkým počtem prvků s organizovanými interakcemi (i přesto, že tyto interakce jednotlivě nevypadají organizovaně). Jelikož je počet prvků velký (což by vedlo k velkému počtu rovnic při řešení) a interakce jsou organizované a ne nahodilé, nelze použít statistické metody. Pro tyto systémy je vhodné využít modelování a simulaci. Nemusíme sice dostat jasné závěry, ale modelování a simulace nám alespoň pomůže lépe tyto systémy pochopit a přemýšlet o nich.

3.3 Charakteristiky komplexních systémů

Komplexní systémy si přiblížíme pomocí tzv. typických charakteristik. Patří mezi ně: **složité vazby a nelinearita, samo-organizace a emergentní chování, dynamičnost a adaptibilita**.

V komplexním systému můžeme říci, že „vše souvisí se vším“. Mezi jednotlivými prvky systému existuje množství různých **vazeb**, které mohou být i obousměrné. Tyto vazby tvoří tzv. zpětné vazby. Těžko rozpoznáme příčiny a následky, změny ovlivňují samy sebe. U lineárních systémů je



dopad zásahu přímo úměrný zásahu samotnému. U komplexních systémů mohou mít velké zásahy malý vliv a malé zásahy velký vliv. Např. pohyb planet lze spočítat na stovky let dopředu, naproti tomu počasí na Zemi je velmi obtížné spočítat na příští den. U nelineárních systémů tedy nepřesnost předpovědi vždy rychle narůstá.

Díky zpětným vazbám vzniká v komplexních systémech **řád** vycházející z malých a náhodných impulzů, kdy neexistuje žádný vnější zdroj onoho řádu. V těchto případech vzniká řád tzv. **samoorganizací**. Obecně je vcelku obtížné definovat pojem samoorganizace systému. Vše záleží na typu systému a také na úhlu pohledu pozorovatele. Systém můžeme např. popsat jako samoorganizující v případě, že jeho komponenty spolu interagují tak, aby dosáhly globální požadované funkce či chování, apod. Např. u instituce, školy, jsme schopni popsat, proč daná instituce vykazuje řád. U školy je zdrojem řádu rektor, případně vedoucí fakult a kateder. U mravenčího společenství těžko hledáme zdroj onoho řádu. Zde vzniká řád samoorganizací. Další zajímavou vlastností komplexních systémů je tzv. **emergence**. Určitě jste slyšeli rčení, že celek je více, než součet jeho částí. Právě toto můžeme označit za emergentní chování. Pokud bychom vzali jednotlivé samostatné prvky systému, tak jednotlivě by vůbec nevykazovaly chování, které jsme schopni pozorovat na úrovni celku. Je to proto, že chování celku není prostým součtem jeho částí, ale v úvahu musíme brát jednotlivé vazby mezi jednotlivými prvky. Kdybychom např. odebrali z mraveniště jednoho mravence, těžko zjistíme něco o tom, jak funguje společenství mravenců. Naproti tomu jednoduchý systém můžeme rozložit na jednotlivé části, kdy celek je součtem těchto částí. To je příklad třeba plynů, kdy i na základě malého vzorku plynu jsme schopni vcelku přesně usoudit něco o vlastnostech celku (teplota, hustota).

Samozřejmě, těžko se hledá hranice mezi jednoduchými, neorganizovanými a komplexními systémy. Jeden z pohledů říká, že existují jen komplexní systémy. Jednoduché a neorganizované, resp. lineární systémy jsou jen našim zjednodušením těchto komplexních systémů. Je tu jistá analogie např. s fyzikou. Pro malé rychlosti a složitosti si bez problémů vystačíme s newtonovskou fyzikou. Se vztušujícími rychlostmi bližící se rychlosti světla ovšem už s newtonovskou fyzikou nevystačíme a musíme zapojit fyziku relativistickou.

V komplexních systémech neustále dochází ke změnám. Tyto systémy jsou neustále v pohybu, a pokud se nám zdá, že jsou v klidu, je to jenom dočasně. Často můžeme v komplexních systémech pozorovat **období klidu** střídající **období velkých změn**. Můžeme to vidět např. na historii lidstva. Průběhu známé historie vzniklo a zaniklo množství různých útvarů, říší, států a jiným celků. Tyto útvary měly své období stability. K zániku, případně transformaci těchto útvarů potom dochází relativně rychle vzhledem k období stability. Schopnost adaptace je také důležitou vlastností těchto systémů. Komplexní systémy bývají většinou heterogenní, kdy jednotlivé části mají různé „schopnosti“, které díky samoorganizaci a emergenci dokáží přestát různé vnější i vnitřní podněty a adaptovat své chování na aktuální situaci. Příklad můžeme vzít opět z lidské historie. V průběhu staletí vznikla spousta vesnic a měst, jednoduše řečeno míst, kde lidé žijí, pracují a odpočívají. V průběhu času některá města zanikla, vesnice



Řád v komplexním systému



Uvažování o systémech

Období klidu

Období změn



se rozrostly a staly se z nich velké aglomerace. Vše díky **adaptaci**, která se projevuje nejen na úrovni částí, ale i systému jako celku. Na úrovni částí jde o jednotlivé osoby, které se stěhují z jednoho místa na místo, a mění zaměstnání dle poptávky. Na úrovni celku pak celý systém „přežije“ a adaptuje se, i když některé prvky zcela zaniknou.

3.4 Studium komplexních systémů



Komplexní systémy jsou opravdu **složité** a **neintuitivní**, celek je více než prostý součet všech částí. Přes to všechno jsme komplexní systémy schopni studovat. Před samotným studiem těchto systémů je nutné si uvědomit několik principů. Na komplexní systémy bychom se měli dívat z nadhledu, což vychází právě z toho, že celek je více než součet jeho částí. Nebudeme tedy zkoumat systém po jeho prvcích, ale z celkového pohledu. Při zkoumání vždy postupujeme od obecnějšího ke specifickějšímu. Komplexní systémy jsou **dynamické**, tudíž věnujeme pozornost dynamice jeho procesů. Pozornost věnujeme vztahům a interakcím mezi prvky systému. Důležité je vzít v potaz vliv pozorovatele. Každý pozorovatel má jiný úhel pohledu, dokonce někdy dokáže svým pozorováním chování daného systému ovlivnit. U komplexních systémů jde o běh na dlouho trať, vždy je zkoumáme v delším časovém horizontu. Pokud bychom jej zkoumali v krátkém horizontu, mohli bychom o jeho chování zjistit naprosto mylné informace.



Pro ilustraci toho, jak těžké a nepoužitelné může být usuzování o celku, když zkoumáme jen část, je starověké podobenství o slepých mužích. Šest slepých mužů se vydalo hledat slona, aby zjistili, jak vypadá. Když slona najdou, každý z šesti mužů nahmatá jinou část slona a popisuje slona ostatním. První muž nahmatá chobot a říká: „Slon vypadá jako had.“ Druhý muž nahmatá nohu a říká: „Slon vypadá jako kmen stromu.“ Třetí muž nahmatá ocas a říká: „Slon vypadá jako provaz.“ Čtvrtý muž nahmatá kel a říká: „Slon vypadá jako kopí.“ Pátý muž nahmatá ucho a říká: „Slon vypadá jako vějíř.“ Šestý muž nahmatá bok a říká: „Slon vypadá jako zed.“ Každý z těch mužů má pravdu. Ale slon nevypadá ani jako jedno z přirovnání. Zde vidíme nebezpečí, chceme-li usuzovat o celku z moc jemných detailů a zároveň nám toto přirovnání ilustruje i relativnost úhlu pohledu.

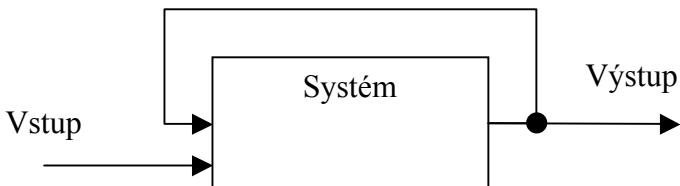
Lineární uvažování



Dalším úskalím při studiu komplexních systémů je **lineární uvažování** s krátkodobým výhledem a zjednodušené uvažování o příčinách. Lidé často mají tendenci uvažovat lineárně i o komplexních systémech. To ilustruje známý příklad s rybníkem a řasami. Správce rybníka zpozoruje, že se na hladině rybníka začínají tvořit řasy. Jejich počet se každý den zdvojnásobí. Ze začátku správce problém dlouhou dobu neřeší, jelikož vypadá, že řasy přibývají pomalu a stále je čas problém řešit. Správce začne situaci řešit, až když je polovina hladiny rybníka pokrytá řasami. To už na řešení problému zbývá už jen jeden den. Chybí zde dlouhodobý výhled. Charakteristickým rysem komplexních systémů je, že změny s dlouhodobým pozitivním efektem vedou často ke krátkodobým zhoršením. Bez dlouhodobého výhledu, bychom nikdy nebyli schopni vidět tyto pozitivní změny. Proto je často velmi složité prosadit některá politická či ekonomická rozhodnutí, která mají krátkodobý negativní efekt.

3.5 Zpětná vazba

Zpětné vazby jsou nedílnou součástí komplexních systémů. Můžeme si je představit jako **zpětnovazební smyčku**, které z výstupu systému vedou zpět na jeho vstup (Obrázek 4).



Obrázek 4 – Schéma zpětné vazby

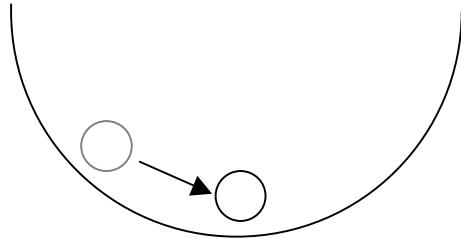
Takovou zpětnou vazbu můžeme najít nejen v oboru modelování a simulace, ale také např. v teorii řízení. Toto schéma může představovat také biologický systém a látkový tok, elektromagnetický signál v technice aj.

Zpětnou vazbu dělíme na **negativní** a **pozitivní**. Negativní zpětná vazba udržuje systém **v rovnováze**, zatímco pozitivní zpětná vazba jej **z rovnováhy vyvádí**.

Klasickým příkladem pro demonstraci zpětných vazeb je model dravec-kořist. Kořist může být reprezentována např. populací ovcí a dravcem populací vlků. Za pozitivní zpětnou vazbu můžeme považovat samotné rozmnožování ovcí. Čím více ovcí, tím více se jich rozmnoží a stále dokola. Samozřejmě, čím více ovcí, tím více se rozmnoží i populace vlků. Zatímco pozitivní zpětná vazba vyvádí systém z rovnováhy tím, že způsobuje nekontrolovatelný růst populace ovcí, negativní zpětná vazba ve formě vlků požírajících ovce má na tento systém stabilizující vliv. Efekty negativní a pozitivní vazby nelze jednoduše odhadnout, velkým vliv na efekty zpětných vazeb mají zpoždění zpětných vazeb. Díky tomu také populace ovcí a vlků v modelu dravec-kořist často osciluje okolo určité hodnoty.

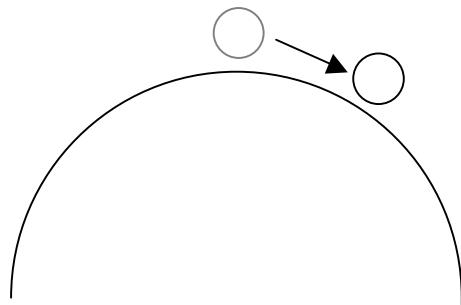
Negativní zpětná vazba znamená, že změna, která se projevila na výstupu, vede v konečném důsledku ke zmenšení této změny (Obrázek 5). Negativní zpětné vazby mají regulační charakter. Ne nadarmo se efektu negativní zpětné vazby používá při regulaci teploty v místnosti. Je-li v místnosti příliš teplo, řídící jednotka vypne topení. Je-li naopak příliš chladno, topení je zapnuto. Díky této zpětné vazbě (regulační smyčce) je udržována žádaná hodnota teploty v místnosti v rovnováze (s mírnými oscilacemi způsobenými zpožděním). V příkladu s vytápěním máme řídící jednotku, která je centrem řízení vydávající pokyny o efektu zpětné vazby. Zpětné vazby ovšem fungují i bez centrálního řízení. Příkladem je právě prostředí burzy, který jsme si uvedli na začátku kapitoly o komplexních systémech.





Obrázek 5 – Negativní zpětná vazba

Pozitivní zpětná vazba znamená, že změna, která se projevila na výstupu, vede v konečném důsledku ke zvětšení této změny (Obrázek 6). Pozitivní zpětné vazby mají deregulační charakter. Efekty pozitivní zpětné vazby mohou mít jak destruktivní charakter, tak můžou přispět ke vzniku nových struktur. Příkladem destruktivního efektu pozitivní zpětné vazby je jaderná exploze. Při rozštěpení jednoho atomu uranu se uvolní dva až tři neutrony, která jsou schopny způsobit štěpnou reakci v dalších atomech uranu, až dojde k jaderné explozi. Příkladem pozitivní zpětné vazby může být vznik měst. Na příhodném místě u soutoku řek a blízké obchodní cesty se usadí několik rodin. Díky příhodným podmírkám a rozvíjejícímu se obchodu se zde usadí více a více nových přistěhovalců až původně místo o několika chalupách vyroste ve velkou metropoli.



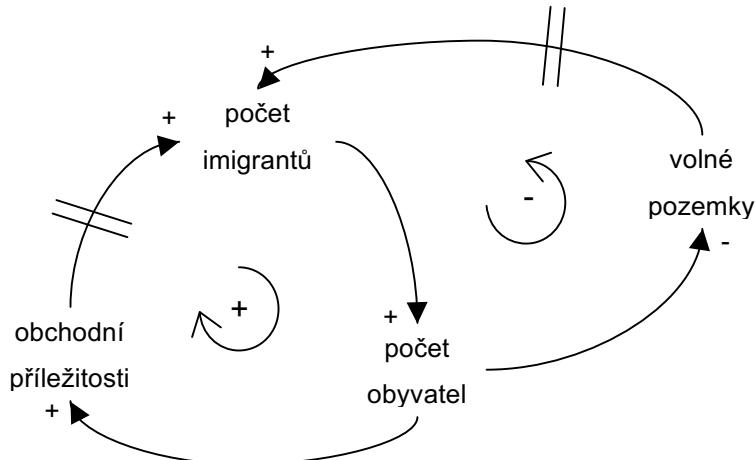
Obrázek 6 - Pozitivní zpětná vazba

Negativní a pozitivní zpětná vazba se nikdy nevyskytují samostatně. V každém systému můžeme najít jak pozitivní, tak negativní zpětnou vazbu. Zpětnovazebních smyček je zde většinou několik a vytvářejí komplikované zpětnovazební cykly. Tak jako třeba při pozitivní zpětné vazbě způsobující jadernou explozi se projeví negativní zpětná vazba představující vyčerpání štěpného materiálu. Tak stejně populace rozrůstající se metropole narazí na efekt negativní zpětné vazby představující zbylé volné pozemky, anebo množství potravin schopné uživit tak velkou populaci.

Pro porozumění, které zpětné vazby se v systému vyskytují, nám pomůže **diagram struktury zpětnovazebních cyklů**, nazývaný také diagram kauzálních smyček. Ten vytvoříme tak, že si zakreslíme klíčové položky systému a mezi jednotlivými položkami zaznačíme vztah příčina-následek pomocí šipky, existuje-li. Znaménkem (+) značíme pozitivní vztah, znaménkem (-) vztah negativní. Každý takto vytvořený cyklus označíme jako pozitivní či negativní. Cyklus označíme jako pozitivní (negativní) zpětnou vazbu, pokud obsahuje sudý (lýchý) počet negativních vztahů. Šipka může také obsahovat symbol (||) značící, že mezi příčinou a následkem



existuje znatelné zpoždění. Abychom správně přečetli zpětnovazební smyčku, je nutné vybrat nějakou startovní proměnnou, od které začneme čist a vybereme směr – více/méně. Více *populace* vede k větší *porodnosti* a to vede k větší *populaci*. Méně *populace* vede k menší *porodnosti* a to vede k menší *populaci*. Toto je příklad pozitivního cyklu (anglicky reinforcing feedback loop). Pozitivní cyklus se projeví stejně v obou směrech. Negativní cyklus je na to opačně. Více *populace* vede k vyššímu počtu *úmrtí*, vyšší počet *úmrtí* vede k menší *populaci*. Vidíme, že negativní cyklus obsahuje lichý (jeden) počet negativních vztahů (počet *úmrtí* vs. *populace*).



Obrázek 7 - Diagram struktury zpětnovazebních cyklů metropole



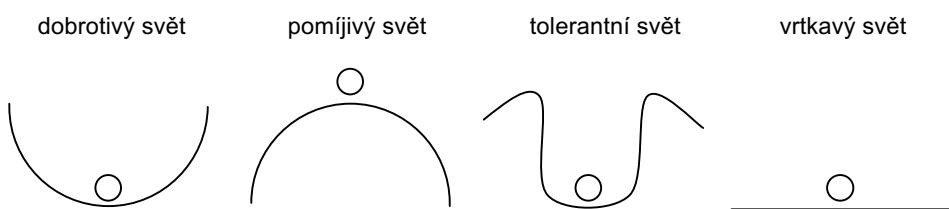
Pojďme si vysvětlit diagram struktury zpětnovazebních cyklů na konkrétním příkladu (Obrázek 7). Je to již zmiňovaný scénář vzniku měst. Je zde několik proměnných, které byly vybrány jako důležité pro to, co chceme daným diagramem vyjádřit. Začněme proměnnou *počet obyvatel*. Více *obyvatel* vede k většímu počtu *obchodních příležitostí* (+). Více *obchodních příležitostí* vede k většímu počtu *imigrantů* (+). Více *imigrantů* vede k většímu počtu *obyvatel*. Všechny vztahy jsou pozitivní, tudíž tento cyklus označíme také jako pozitivní. V konečném důsledku to znamená, že více/méně obyvatel vede opět k více/méně obyvatel. Cyklus napravo také začněme od *obyvatel*. Více *obyvatel* vede k menšímu počtu *volných pozemků*, tj. opačný směr více vs. méně, proto negativní vztah (-). Méně *volných pozemků* vede k menšímu počtu *imigrantů* (+). Méně *imigrantů* vede k menšímu počtu *obyvatel* (+). Stejného výsledku bychom dosáhli, začali bychom číst: méně *obyvatel* vede k většímu počtu *volných pozemků* (-). V tomto cyklu se tedy nachází lichý počet (jeden) negativní vztah. Z tohoto důvodu označíme tento cyklus jako negativní, jelikož více/méně *obyvatel* vede v konečném důsledku k méně/více *obyvatel*.

Mezi *obchodními příležitostmi* a počtem *imigrantů* je symbol (||) značící, že se zde vyskytuje větší zpoždění, které chceme zdůraznit. Chvíli totiž trvá, než se potenciální *imigranti* dozvědí o nových *obchodních příležitostech* a než se rozhodnou nastěhovat se do nového města. Obdobně nalezneme zpoždění i mezi *volnými pozemky* a *počtu imigrantů*. Tato zpoždění s velkou pravděpodobností povedou k oscilaci populace rozrůstající se metropole.

Povaha světa dle Adamse

Tento příklad zcela nepopisuje všechny aspekty rozrůstající se metropole. Nebere například v úvahu úmrtnost obyvatel. Vždy ovšem záleží na tom, které faktory jsou pro nás z hlediska zkoumání důležité.

Důsledkem protikladných sil zpětných vazeb může být stav, kdy je systém v rovnováze. I když slovo rovnováha může navozovat pocit, že se v systému nic neděje, i drobný impulz může vyvést systém z rovnováhy. Rovnováha může být jak vysoce stabilní tak nestabilní. Několik pohledů na rovnováhu nabízí obrázek, který přestavuje čtyři pohledy na povahu světa podle J. Adamse (Obrázek 8).



Obrázek 8 - Dobrotivý svět je předpověditeLNý, robustní, stabilní, v systémech dominují negativní zpětné vazby.

Pomíjivý svět je křehký, nestabilní, změny jsou nevratné, v systémech dominují pozitivní zpětné vazby.

Tolerantní svět je kombinací předchozích dvou. Do určitých limitů je stabilní a předpověditeLNý, po překročení těchto limitů se stává nestabilní.

Vrtkavý svět je nepředpověditeLNý a chová se náhodně.

Kontrolní otázky:

1. Co je to komplexní systém?
2. Jaké druhy systémů rozeznáváme a podle jakých kritérií jsme je schopno rozdělit?
3. Jaká úskalí nás čekají při studiu komplexních systémů?
4. Vysvětlete fungování zpětnovazebních cyklů.



Korespondenční úkol:

Vyjděte z diagramu struktury zpětnovazebních cyklů metropole a přidejte alespoň dva zpětnovazební cykly. Můžete zohlednit např. úmrtnost populace, nemoci, potraviny, zábavu, těžbu surovin, řemesla aj.



Úkoly k zamýšlení:

1. Zkuste zapárat v historii naší civilizace a najít případ kdy malá změna způsobila ohromnou změnu v uspořádání světa a kdy naopak velká změna nezpůsobila nějaké znatelné změny v uspořádání. Můžete zvážit i hypotetické scénáře.



Shrnutí obsahu kapitoly

V této kapitole jsme se seznámili s komplexními systémy. Vysvětlili jsme si, jak je takový systém charakterizován a na co musíme pamatovat při jeho

studiu. Představili jsme si také několik vlastností komplexního systému jako např. emergence nebo samoorganizace. Zmínili jsme také některá úskalí při studiu těchto systémů a to zejména usuzování o celku z jednotlivých částí a lineární uvažování. Nakonec jsme si přestavili diagram zpětnovazebních cyklů a rozebrali fungování zpětnovazebních smyček.

4 Matematické prostředky pro modelování a simulaci

V této kapitole se dozvítíte:

- Matematické prostředky
- Matematické metody
- Fáze procesu simulace systému

Po jejím prostudování byste měli být schopni:

- získali základní přehled o matematických prostředcích a metodách používaných nejčastěji při modelování a simulaci,
- dokázali vysvětlit základní fáze procesu simulace systému.

Klíčová slova této kapitoly:

Základní pojmy, modelování, simulace, model.

Doba potřebná ke studiu: 4 hodiny

Průvodce studiem

V této části se seznámíte se základními prostředky a metodami používanými v oblasti matematického modelování a simulace systémů, ať už diskrétních nebo spojitých. Uvedený přehled matematických metod a teorií užívaných v oblasti modelování a simulace nemůže být přirozeně úplný. Výběr těchto metod je do značné míry subjektivní, ovlivněný zaměřením původních autorů.



4.1 Matematické prostředky

Uvedeme jen některé významnější matematické prostředky používané při vytváření modelů. Tyto prostředky patří k nejvíce používaným a nalezneme je zejména v přístupu shora dolů, ovšem některé z nich se uplatní také u přístupu zdola nahoru (např. maticová algebra).

Teorie množin a transformací se užívá především k popisu změn stavu systému. Předpokládejme, že jsme definovali množinu, jejíž prvky reprezentují všechny možné stavy systému, které se mohou realizovat v průběhu jeho vývoje. Tato množina nemusí být přirozeně konečná, jejími prvky mohou být např. písmena nějaké abecedy. Dále zvolíme vhodný časový interval a ke každému ze stavů přiřadíme stav, do něhož by uvažovaný systém v průběhu zvoleného časového intervalu mohl přejít. Směr přechodu znázorníme šipkou. Takto vznikne model stavových změn (stavových přechodů), jenž kvalitativně popisuje vývoj sledovaného systému (viz schéma na Obrázek 9).



Obrázek 9 – Model stavových změn

Uvedené schéma můžeme též považovat za orientovaný graf, jehož uzly představují stavy systému a hrany možné stavové přechody. Takové schéma se nazývá **stavovým diagramem**. Typickým představitelem stavového diagramu je např. konečný automat.

Stavovým diagramem je možné popsat také stavy jednotlivých prvků systému a jejich změnu stavu v průběhu vývoje. Tuto situaci popisuje diagram na (Obrázek 10), kde jednotlivé přechody mezi stavy popisují, jaké vývojové stadium následuje zvolenému stavu, resp. proces transformace stavů jednotlivých prvků a generování prvků nových. Není zde tedy žádný počáteční či koncový stav.

Pozn., kvalitativní výzkum se zabývá např. popisem procesů, vzájemných vztahů, situací, systémů, interpretací, zobecnění atd. Naproti tomu kvantitativní metody se nejlépe hodí ke zkoumání jednoduchých a měřitelných atributů.

Člověk se setkává se stále složitějšími objekty. Množství informace potřebné k jejich popisu se přirozeně rovněž zvětšuje. Přesné modely složitých systémů se stávají nezpracovatelnými pomocí konvenčních matematických prostředků, proto je třeba hledat prostředky nové. Příčina tohoto stavu spočívá v tzv. principu inkompatibility. Roste-li složitost systému, klesá naše schopnost formulovat přesné a významné soudy o jeho chování, až je dosaženo hranice, za níž se přesnost a relevantnost prakticky vylučují.

Fuzzy množiny

Vhodným prostředkem pro popis nepřesných (vágních) pojmu je **teorie fuzzy množin**. Nejsme-li schopni přesně vymezit hranice nějaké třídy určené vágním pojmem, pak přiřadíme každému prvku stupeň jeho příslušnosti k dané třídě. Bude-li škála pro tuto míru uspořádaná, pak zřejmě platí následující tvrzení. Čím menší je míra příslušnosti daného prvku k uvažované třídě, tím blíže je tento prvek hranici třídy. Tato míra se nazývá stupeň příslušnosti prvku do uvažované třídy a třída, jejíž každý prvek je charakterizován stupněm příslušnosti do ní, se nazývá fuzzy množina. Teorie fuzzy množin lze využít i při zkoumání reálných systémů. Rozlišují se fuzzy systémy dvojího druhu:

- 1 skutečné fuzzy systémy, jejichž přesný teoretický popis neexistuje;
- 2 systémy, jež jsou natolik složité, že je nejsme schopni klasickými metodami přesně popsat.

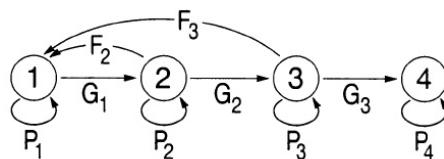
Teorie fuzzy množin vytváří jakýsi most mezi verbálním a matematickým modelem reálného systému.

Lineární algebra

Významnou roli při vytváření modelů hraje **lineární algebra**, zejména **maticová algebra**. Matic se využívá především k matematickému popisu struktury systému a interakcí mezi jednotlivými prvky systému. Např. velikosti populací v n -složkovém systému můžeme popsat pomocí sloupcového vektoru $(x_1, x_2, \dots, x_n)^T$ nebo řádkového vektoru (x_1, x_2, \dots, x_n) , kde x_i značí velikost i -té populace. Interakce mezi jednotlivými populacemi se přehledně vyjadřují pomocí interakční matic (tabulky),

$$\begin{pmatrix} \alpha_{11} & \alpha_{12} & \alpha_{1n} \\ \alpha_{21} & \alpha_{22} & \alpha_{2n} \\ \alpha_{n1} & \alpha_{n2} & \alpha_{nn} \end{pmatrix}$$

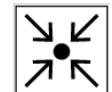
v níž prvek α_{ij} ($i, j = 1, 2, \dots, n$) reprezentuje interakci mezi i -tou a j -tou populací. Maticová reprezentace je typická pro diskrétní modely rozvoje populace. Známá je tzv. Leslieho matice, modelující vývoj organismů rozdělených do věkových tříd stejné délky, jejíž natalita (porodnost) i mortalita (úmrtnost) jsou funkcí věku organismů. Z Leslieho matice vychází Lefkovitchova matice (Obrázek 10), která dokáže modelovat nestejně dlouhé vývojové fáze (ontogeneze) organismu.



$$\begin{pmatrix} 0 & F_2 & F_3 & 0 \\ G_1 & P_2 & 0 & 0 \\ 0 & G_2 & P_3 & 0 \\ 0 & 0 & G_3 & P_4 \end{pmatrix}$$

Obrázek 10 - Stavový diagram a Lefkovitchova matice pro vývoj populace velryb
(http://mathbio.colorado.edu/mediawiki/images/Pro3_fig1.jpg)

Matrice na (Obrázek 10) modeluje 4 vývojová stádia populace velryb, 1 – jednoroček, 2 – mladistvý, 3 – reprodukční věk, 4 – post-reprodukční věk. Koeficient P_1 je v matici nulový, jelikož délka projekce se rovná délce trvání fáze pro jednoročky (1 rok). Označení koeficientů tvořících prvky matice má následující smysl: P – pravděpodobnost setrvání v daném vývojovém stádiu, G – pravděpodobnost přežití a přesunu do dalšího stádia, F – (fertility) násobek počtu potomků z daného stadia. Pozn. někdy se koeficienty P a G souhrnně označují jako P . U Leslieho matice se koeficienty označují písmeny F – (fecundity) potence pro reprodukci a S – (survives) pravděpodobnost přežití a přesunu do dalšího stádia. Leslieho matice nemá oproti Lefkovitchově matici prvky na diagonále.



$$\begin{bmatrix} p_1 \\ \vdots \\ p_n \end{bmatrix}_{t_{n+1}} = \begin{bmatrix} P_{11} + F_{11} & \cdots & F_{n1} \\ \vdots & \ddots & \vdots \\ P_{n1} & \cdots & P_{nn} \end{bmatrix} \times \begin{bmatrix} p_1 \\ \vdots \\ p_n \end{bmatrix}_{t_n}$$

Vektor populace v následujícím kroku se vypočte vynásobením matice se sloupcovým vektorem aktuální populace. Koeficienty matice nemusí být pevné, ale mohou se v čase měnit.

Při konstrukci modelů se nejčastěji uplatňují **diferenční** nebo **diferenciální rovnice**. Pomocí těchto rovnic se simulují časové změny stavových proměnných systému. **Diferenční rovnice** reprezentují změny, které se uskutečňují v průběhu diskrétních časových úseků. Je-li V_t hodnota určité stavové proměnné V v čase t , pak diferenční rovnice

Diferenční rovnice

$$V_{t+1} = f(V_t, t)$$

určuje hodnotu této proměnné (jako funkci původní hodnoty V_t a času t) po uplynutí časové jednotky. Pokud se dynamika systému popisuje soustavou diferenčních rovnic, V_t představuje vektor stavových proměnných v čase t a $f(V_t, t)$ onu soustavu rovnic. Diferenční rovnice jsou mimořádně vhodné pro vyjádření časových zpoždění, např.

$$V_{t+1} = f(V_t, V_{t-1}, \dots, t).$$

Diferenciální rovnice

Diferenciální rovnice popisují změny, které probíhají v čase spojitě, popř. kvazispojitě. Mají obvykle tvar

$$\frac{dV}{dt} = f(V, t),$$

přičemž V , $\frac{dV}{dt}$ i $f(V, t)$ mohou být chápány jako vektory. Diferenciální rovnice uvedeného typu vyjadřuje rychlosť změny stavové proměnné V jako funkci okamžitých hodnot této stavové proměnné a času.



Vedle obyčejných diferenciálních rovnic a jejich soustav se při modelování systémů poměrně často setkáváme i s **parciálními diferenciálními rovnicemi** a jejich soustavami (např. při studiu dynamiky populací v nehomogenním prostředí).

V případě stochastických modelů se využívá **stochastických diferenciálních rovnic**, jež mají tvar

$$dV_t = f(V_t, t) + g(V_t, t) dW_t,$$

v němž $\{V_t, t \geq 0\}$ značí náhodný proces a dW_t diferenciál Wienerova procesu (zjednodušeně náhodné veličiny s normálním rozdělením, jejíž střední hodnota je nulová a rozptyl roven dt). Stochastické diferenciální rovnice se uplatňují např. v genetice a při modelování aktivity neuronů.

Kromě již zmíněných matematických prostředků se při modelování systémů často uplatňují:

- matematická logika (klasická logika, vícehodnotová logika, fuzzy logika, temporální logika);
- integrodiferenciální a integrální rovnice (např. při studiu dynamiky populaci se započtením zpoždění ve vzájemných interakcích);
- orientované grafy (při modelování transportních jevů); strukturní termodynamika (zobecněné termodynamické síly a toky při modelování transportu).

4.2 Matematické metody

V této části naleznete přehled a stručnou charakteristiku vybraných matematických metod, které vycházejí z konzistentních teoretických

základů. Pamatujte si, že metoda se opírá také o teorii, je tedy „něco více“ než pouhý prostředek.

Pohybové rovnice systému mají často velmi jednoduchý tvar

Pohybové rovnice

$$\frac{d\mathbf{V}}{dt} = \mathbf{f}(\mathbf{V}), \quad (4.1)$$

kde \mathbf{V} značí vektor stavových proměnných délky n a \mathbf{f} zobrazení $\mathbf{R}_n \rightarrow \mathbf{R}_n$. Řešením této rovnice na časovém intervalu T rozumíme zobrazení $\varphi: T \rightarrow \mathbf{R}_n$ takové, že pro všechna $t \in T$ platí

$$\frac{d\varphi(t)}{dt} = \mathbf{f}(\varphi(t)).$$

Dále se předpokládá, že každým bodem stavového prostoru prochází právě jedno řešení a že každé řešení je možno prodloužit na interval $(-\infty, \infty)$. Klasický přístup vyžaduje nalezení obecného řešení rovnice (4.1) ve tvaru $\varphi(t, C_1, C_2, \dots, C_n)$, přičemž hodnoty konstant C_1, C_2, \dots, C_n se stanoví z počátečních podmínek. Získáme-li obecné řešení, máme úplný obraz o všech řešeních rovnice (4.1). Ze zkušenosti však víme, že (s výjimkou lineárních rovnic) existují jen velmi speciální třídy diferenciálních rovnic, jejichž obecné řešení lze rozumně vyjádřit.

Při studiu dynamiky systému popsaného rovnicí (4.1) nepotřebujeme zpravidla znát obecné řešení, úplně postačí informace kvalitativní (resp. popisné) povahy o chování systému v tzv. ustálených režimech. Informace tohoto typu poskytuje **kvalitativní teorie řešení diferenciálních rovnic**.

Zavádí se pojem dynamického systému jako zobrazení

Kvalitativní teorie
diferenciálních rovnic

$$\phi: \mathbf{R} \times \mathbf{R}_n \rightarrow \mathbf{R}_n, \quad \phi(t, V) = \phi_t(V),$$



které má následující vlastnosti:

- ϕ_0 je identita ($\phi_0(V) = V$);
- ϕ je pro každé $t \in \mathbf{R}$ difeomorfismus (homeomorfismus, jemuž přísluší diferencovatelné inverzní zobrazení);
- pro všechna $t, s \in \mathbf{R}$ platí

$$\phi_{t+s}(V) = \phi_s(\phi_t(V)).$$

Ze základních vět o spojitosti a diferencovatelnosti řešení diferenciální rovnice (4.1) vyplývá, že každá taková diferenciální rovnice generuje dynamický systém, v němž $\phi_t(V)$ představuje řešení této rovnice v čase t za předpokladu, že toto řešení vychází (v čase $t = 0$) ze stavu V . **Trajektorií (orbitou)** vycházející ze stavu V pak rozumíme množinu $\{\phi_t(V); t \in \mathbf{R}\}$.

Vzhledem k tomu, že vektorová funkce \mathbf{f} v rovnici (4.1) nezávisí explicitně na čase t , nezáleží vůbec na tom, ve kterém časovém okamžiku

prochází řešení této rovnice stavem V . Zajímají nás jen takové případy, kdy každým bodem $\mathbf{V} \in \mathbf{R}_n$ prochází právě jediná orbita. Orbita může být:

- 1 bodem, jestliže platí $\mathbf{f}(\mathbf{V}) = 0$ (takový bod se nazývá **kritický bod**);
- 2 diferencovatelnou křivkou, která je uzavřená právě tehdy, když je řešení rovnice (4.1) procházející bodem V periodické.



Kritické body se interpretují jako rovnovážné stavy dynamického systému, kritické body spolu s uzavřenými křivkami (periodickými orbitami) jako **ustálené režimy** tohoto systému.

Kvalitativní teorie diferenciálních rovnic hledá především odpovědi na následující otázky.

- 1 Jaké jsou ustálené režimy sledovaného dynamického systému?
- 2 Je ustálený režim stabilní? Vrátí se systém po vychýlení z ustáleného režimu zpět nebo přejde do nějakého jiného ustáleného režimu nebo bude „bloudit“, aniž by se vůbec dostal do nějakého ustáleného režimu?



Všechny otázky kvalitativní povahy lze uspokojivě zodpovědět v případě, že známe rozložení orbit dynamického systému (tzv. **fázový portrét systému**). Představu o kvalitativní shodě fázových portrétů dvou dynamických systémů vyjadřuje nejlépe pojem topologické ekvivalence dynamických systémů. Dva dynamické systémy ϕ, ψ na množině \mathbf{R}_n se nazývají **topologicky ekvivalentní**, existuje-li homeomorfismus $h: \mathbf{R}_n \rightarrow \mathbf{R}_n$, jenž zobrazuje orbity systému ϕ na orbity systému ψ . Kvalitativním řešením rovnice (4.1) rozumíme nalezení topologické struktury této rovnice, tj. zařazení rovnice do příslušné třídy topologické ekvivalence. Lokální topologická klasifikace diferenciálních rovnic typu (4.1) v okolí kritických bodů i v okolí periodických orbit je uspokojivě vyřešena. Prakticky všechny možné diferenciální rovnice tohoto typu je možno zařadit do konečného počtu tříd, přičemž kritérium pro jejich zařazení je relativně jednoduché. Problém globální topologické klasifikace zůstal dosud nevyřešen. Doporučuje se provádět topologickou klasifikaci jen pro tzv. strukturně stabilní diferenciální rovnice. Rovnice (4.1) se nazývá strukturně stabilní, jestliže všechny rovnice z nějakého jejího okolí jsou s ní topologicky ekvivalentní. Nepřesnosti v zadání takové rovnice pak nemohou změnit kvalitativní chování příslušného dynamického systému.

V rámci kvalitativní teorie řešení diferenciálních rovnic se studují i rovnice závislé na parametru, tj. rovnice typu

$$\frac{d\mathbf{V}}{dt} = \mathbf{f}(\mathbf{u}, \mathbf{V}), \quad (4.2)$$

kde parametr \mathbf{u} je obecně vektorem délky p a $\mathbf{f}: \mathbf{R}_p \times \mathbf{R}_n \rightarrow \mathbf{R}_n$. Je-li pro $\mathbf{u} = \mathbf{u}_0$ rovnice (4.2) strukturně stabilní, pak pro \mathbf{u} z nějakého okolí \mathbf{u}_0 se topologická struktura této rovnice nezmění. Pokud však rovnice (4.2) není pro \mathbf{u}_0 strukturně stabilní, existují v libovolném okolí \mathbf{u}_0 rovnice typu (4.2) s různými topologickými strukturami. Taková změna topologické struktury rovnice (4.2) v důsledku změny parametru \mathbf{u} se nazývá **bifurkace**. Problematice bifurkací je věnován přehledný článek.

Při modelování a simulaci některých jevů (např. šíření nervového vztahu, psychické jevy) se s úspěchem používá **Thomovy teorie katastrof**. K základním pojmu teorie katastrof lze dospět řešením soustav diferenciálních rovnic typu (4.2), kde vektorový parametr **u** popisuje vnější podmínky. Podobně jako v kvalitativní teorii řešení diferenciálních rovnic nás zajímají především ustálené režimy systému. Chování systému popsaného soustavou diferenciálních rovnic (4.2) je přirozeně závislé na hodnotě parametru **u**. Při změně tohoto parametru dochází za jistých okolností v některých bodech, jímž se říká katastrofické body, k náhlé změně ustáleného režimu systému. Tyto náhlé změny (skoky) se nazývají katastrofami. Teorie katastrof se (řečeno zjednodušeně) zabývá řešením pohybových rovnic (4.2), ustálenými režimy vektoru **V** stavových proměnných, závislostí těchto ustálených režimů na hodnotách parametru **u** a zejména způsoby, jakými se může ustálený režim systému v katastrofických bodech měnit. Podstatnou kapitolu teorie katastrof představuje právě klasifikace těchto způsobů skokových změn (klasifikace katastrofických bodů). Z teorie katastrof je nejvýznamnější tzv. elementární teorie katastrof, jež zkoumá speciální (gradientový) případ, kdy

$$f_i(u, V_1, V_2, \dots, V_n) = \frac{\delta P}{\delta V_i}$$

a P je nějaká reálná funkce definovaná na množině $\mathbf{R}_p \times \mathbf{R}_n$.

Zatímco teorie katastrof jako matematická disciplína je nepochybně významným oborem, její aplikace (metoda katastrof) je předmětem četných sporů. Používání pojmu a výsledků teorie katastrof má do značné míry heuristickou povahu. Všeobecně se soudí, že při seriózním modelování či simulaci je žádoucí kombinovat metodu katastrof s jinými matematickými prostředky.

K modelování a simulaci spojitéch transportních jevů se často využívá **metody kompartmentových modelů (kompartmentových systémů)**. Klasické kompartmentové systémy je možno považovat za modely systémů hydrodynamických. Představují systém idealizovaných nádob (tzv. **kompartmentů**), jimiž proudí sledované látky (zpravidla směs nosiče a stopovací látky). Existují přirozeně i kanály, kterými látka proudí z okolí systému do některých kompartmentů (vstupy) nebo kterými opouští systém (výstupy). Objem kanálů se zanedbává, přitom se však předpokládá, že jimi může procházet nenulové množství látky v průběhu konečného (nenulového) časového intervalu. Každá nádoba je charakterizována vektorem atributů, jehož složky udávají množství (objemy) jednotlivých druhů látek a rychlosti jejich časových změn. O obsahu kompartmentů se předpokládá, že je homogenní (dokonale promíchaný). To znamená, že když do nějakého kompartmentu látka vstoupí, pak se okamžitě smísí s tím, co bylo v tomto kompartmentu dříve. Výhoda kompartmentových modelů spočívá v tom, že jsou to modely velmi názorné a jejich dynamiku lze poměrně jednoduše popsat analyticky pomocí nějaké soustavy obyčejných diferenciálních rovnic prvního řádu. Metody kompartmentových modelů se z počátku využívalo zejména v nukleární medicíně a radiobiologii. Později pronikly kompartmentové modely i do jiných oborů, např. do farmakologie

Kompartimentové
modely



a cytologie. Metoda kompartmentových modelů se přirozeně dále rozvíjí a v současné době se používá k modelování a simulaci i v epidemiologii a demografii. Obecná metoda multikompartimentových modelů je podrobně popsána v práci (Kotva 1979), matematizace jejích základních představ a principů je obsahem výzkumné zprávy (Křivý 1986). Existují dokonce specializované simulační programovací jazyky pro analýzu kompartmentových systémů (např. jazyk COSMO (Kindler 1969)).

Metoda GUHA

Pro zpracování velkých objemů dat (získaných např. zjišťováním anamnestických a diagnostických údajů na velkém počtu objektů) je určena **metoda automatizovaného generování hypotéz (metoda GUHA)** (Hájek a Havránek 1978). V tomto případě jde o aplikaci vyjadřovacích a deduktivních prostředků matematické logiky na analýzu empirických dat. Z formálního hlediska se analyzují data, která tvoří matematickou strukturu

$$\langle M, \varphi_1, \varphi_2, \dots, \varphi_n \rangle,$$

v níž M je neprázdná konečná množina objektů a $\varphi_1, \varphi_2, \dots, \varphi_n$ zobrazení typu $\varphi_i: M \rightarrow V_i$, kde V_i jsou množiny hodnot sledovaných vlastností na jednotlivých objektech. Původně byla metoda GUHA navržena pro zpracování dvouhodnotových dat, tj. pro případ $V_i = \{0, 1\}$, $i = 1, 2, \dots, n$. V současné době jsou základní principy i matematická teorie metody GUHA formulovány natolik obecně, že jí lze užívat prakticky pro libovolný typ dat. Základním principem metody GUHA je syntaktické vymezení jisté třídy relevantních otázek, které mohou být na základě analýzy dat jednoznačně zodpovězeny. Tyto otázky jsou pak automaticky postupně generovány a zodpovídány na konkrétních datech. Metoda GUHA se neuplatňuje přímo v etapě vytváření modelu zkoumaného systému, ale především při testování hypotéz o zkoumaném systému. Je užitečná všude tam, kde jde o rozsáhlá experimentální data a jejich orientační analýzu (**exploratory analysis**). Pomocí metody GUHA se automaticky vytvářejí (vyhledávají) hypotézy o zkoumaném systému. Takové situace jsou běžné např. v klinických výzkumech a při komplexním studiu ekosystémů.

Matematická statistika

Významnou roli při studiu systémů hrají **metody matematické statistiky**. Těchto metod se využívá především v etapě ověřování hypotéz o studovaném systému na základě analýzy experimentálních údajů. Existuje přirozeně velmi bohatá a pestrá škála seriózních statistických metod, ne všechny se však uplatňují při verifikaci modelu ve stejné míře. V oblasti modelování a simulace se vedle testování statistických hypotéz nejčastěji uplatňují:

- 1 metody regresní a korelační analýzy,
- 2 analýza časových řad,
- 3 metody vícerozměrné statistické analýzy.

Regresní analýza

Úkolem **regresní analýzy** je nalezení vhodné teoretické regresní funkce k vystižení sledované závislosti, určení bodových, popř. intervalových, odhadů regresních koeficientů, určení odhadů hodnot regresní funkce pro účely prognostické a ověření souladu mezi navrženou regresní funkcí a experimentálními daty. Základním cílem **korelační analýzy** je měřit sílu

Korelační analýza

(intenzitu, těsnost) korelační závislosti mezi sledovanými veličinami. V aplikacích jde většinou o vícenásobnou regresi a korelaci, protože se sleduje závislost vybrané náhodné veličiny na skupině dvou a více jiných veličin.

Problematice **analýzy časové řady** se detailně věnují předměty ANDAT a AVDAT. V praxi se nejčastěji vyžaduje vyrovnání dané časové řady a predikce hodnot sledované proměnné. Je zřejmé, že současné sledování několika časových řad přináší podstatně více informací než studium jediné řady. Vnitřní vazby mezi časovými řadami mohou totiž lépe osvětlit mechanismus vzájemné interakce mezi sledovanými veličinami. Při analýze vícerozměrných časových se řeší zejména dva okruhy problémů:

- posuzování charakteru a stupně závislosti mezi danými reálnými časovými řadami $\{X_t\}$ a $\{Y_t\}$,
- predikce ve vícerozměrných řadách (např. zlepšení předpovědi veličiny X na základě znalosti historie řady $\{Y_t\}$).

Z metod vícerozměrné statistické analýzy se při modelování a simulaci (v etapě ověřování hypotéz o zkoumaném systému) patrně nejvíce uplatňuje **shluková analýza**, která se zabývá klasifikací mnohorozměrných dat v situacích, kdy nemáme k dispozici žádný model, ale jen data samotná. Je dána nějaká množina $\{O_1, O_2, \dots, O_N\}$ objektů, z nichž každý je reprezentován r -ticí hodnot atributů $\{a_{i1}, a_{i2}, \dots, a_{ir}\}$, tj. bodem v r -rozměrném euklidovském prostoru. Úkolem shlukové analýzy je seskupit objekty do n shluků S_1, S_2, \dots, S_n tak, aby objekty patřící do téhož shluku si byly v určitém smyslu blízké, kdežto objekty, jež náležejí různým shlukům, co nejvíce odlišné. Důležitá je především vhodná volba míry podobnosti či nepodobnosti objektů, resp. shluků. Shlukovací procedury jsou v podstatě dvojího druhu: hierarchické a nehierarchické. V případě hierarchických postupů se shluky vytvářejí tak, že se vychází od jednotlivých objektů a ty se pak postupně shlukují na základě vhodné míry podobnosti (vzdálenosti), až se dospeje k požadovanému počtu shluků, resp. ke shluku jedinému. Nehierarchické postupy mají iterační charakter a jsou založeny na optimalizaci předem definovaného funkcionálu kvality rozkladu, který vyjadřuje matematické požadavky na stupeň podobnosti objektů uvnitř shluků, homogenitu rozložení objektů uvnitř shluků, rovnoměrnost rozložení objektů do různých shluků apod. Od hierarchických postupů se zásadně liší v tom, že připouštějí změnu rozmístění objektů do shluků v průběhu shlukování, a tím i opravu špatně zvoleného počátečního rozdělení. Oblast použití shlukové analýzy je mnohem širší, než se původně předpokládalo. Užívá se všude tam, kde jde o redukci dat s minimální ztrátou informace (např. při generování a ověřování hypotéz o zkoumaném systému, predikci založené na seskupování). Typické jsou např. aplikace v lékařské diagnostice (shlukování pacientů do homogenních tříd), ve farmaceutice (klasifikace pokusných zvířat) apod.

V sedmdesátých letech 20. století se začínají rozvíjet **simulační metody založené na teoriích formálních jazyků a automatů**. Na tomto místě uvádíme alespoň dva velmi známé přístupy:

Analýza časových řad

Shluková analýza

- metoda založená na teorii **Lindenmayerových systémů** (**L - systémů**),
- metoda založená na teorii **celulárních (buněčných) automatů**.

Obě uvedené metody původně vznikly s cílem modelovat (simulovat) evoluci mohobuněčných systémů. Vycházejí z následujících předpokladů:

- základní stavební jednotkou organismu je buňka,
- růst organismu probíhá v diskrétních časových okamžicích,
- změna stavu každé buňky v daném okamžiku je určena jejím vlastním aktuálním stavem a interakcí se sousedními buňkami,
- změna stavu celého organismu probíhá jako synchronizovaná změna stavu všech buněk.

Modely založené na agentech

Ve stejné době se začínají rozvíjet také **modely založené na agentech** (ABM, tzv. agent-based models). Základ teorie těchto agentových modelů tvoří právě buněčné automaty a umělá inteligence. V těchto modelech má smysl hovořit o více než jednom agentu, jelikož náš zajímají zejména vzájemné interakce mezi agenty a z toho vyplývající emergentní chování celého multiagentového systému. Agenty jsou v multiagentovém systému koordinovány a mohou spolu komunikovat, případně kooperovat na různých úrovních.

V zásadě rozlišujeme čtyři typy agentů, a to **reaktivní, deliberativní, sociální a hybridní**. Reaktivní agent má nejjednodušší vnitřní architekturu. Akce agenta jsou pouze reakcí na podněty z prostředí. Deliberativní agent si uchovává reprezentaci prostředí a vnitřních stavů a je schopen vytvářet plány pro dosahování svých cílů. Sociální agent dokáže s ostatními sociálními agenty komunikovat ve vyšším komunikačním jazyce (jako například v jazyce KQML). Hybridní agent je potom kombinací agentů předchozích typů. V posledních letech se čím dál více prosazuje pojem **inteligentní** agent (Nahodil 2007). Ten sjednocuje všechny typy výše jmenovaných agentů, kdy agentové technologie jsou přirozeným rozšířením současných přístupů, opírajících se o jednotlivé dílčí komponenty. Inteligentní agent má vlastnosti jako autonomnost, reaktivitu, proaktivní přístup, sociálnost, schopnost spolupráce, schopnost zdůvodnění, adaptivní chování, důvěryhodnost, mobilitu, věrohodnost, benevolenci, racionálnost a učení, resp. adaptaci.

Agenty se z velké části rozhodují na lokální úrovni, kdy každý agent má nastavena základní pravidla chování, které při své činnosti v rámci určeného prostředí aplikuje. Většinou je aplikováno řízení bez centrální autority, v některých případech je možno aplikovat rovněž centrální řízení. Základní filozofií těchto modelů ovšem ve větší míře bývá větší počet jednodušších agentů, než malý počet sofistikovanějších agentů.

Kontrolní otázky a úkoly:

1. Charakterizujte základní matematické prostředky používané při modelování a simulaci systémů



- Charakterizujte vybrané matematické metody v oblasti modelování a simulace systémů.

4.3 Základní fáze simulace

V tomto odstavci se seznámíte s obsahem upřesněné Dohody o chápání pojmu simulace systémů, přijaté na půdě Komitétu aplikované kybernetiky ČSVTS.

Simulace systémů jako specifické formy procesu poznání se využívá při zkoumání i projektování objektů, dále při výuce, výcviku a v jiných případech sdělování poznatků a hypotéz. Předmětem simulace systémů jsou systémy vymezené na objektech poznání a jejich dynamika ve smyslu jakékoli změny v čase. Simulované systémy mohou být vymezeny jak na objektech již existujících, tak na objektech projektovaných. Připouští se i zkoumání systémů, které nemají bezprostřední vztah k objektivní realitě. Fundamentálním principem simulace systémů je vyvozování soudů o simulovaném systému na základě experimentů s jeho modelem (přesněji simulátorem).



Základní fáze simulace jsou zřejmé z vývojového diagramu na (Obrázek 11).

Vymezením objektu poznání rozumíme vyčlenění zkoumaného objektu z okolního světa, resp. stanovení požadavků na projektovaný objekt a určení použitelných dílčích objektů ke konstrukci projektovaného objektu.

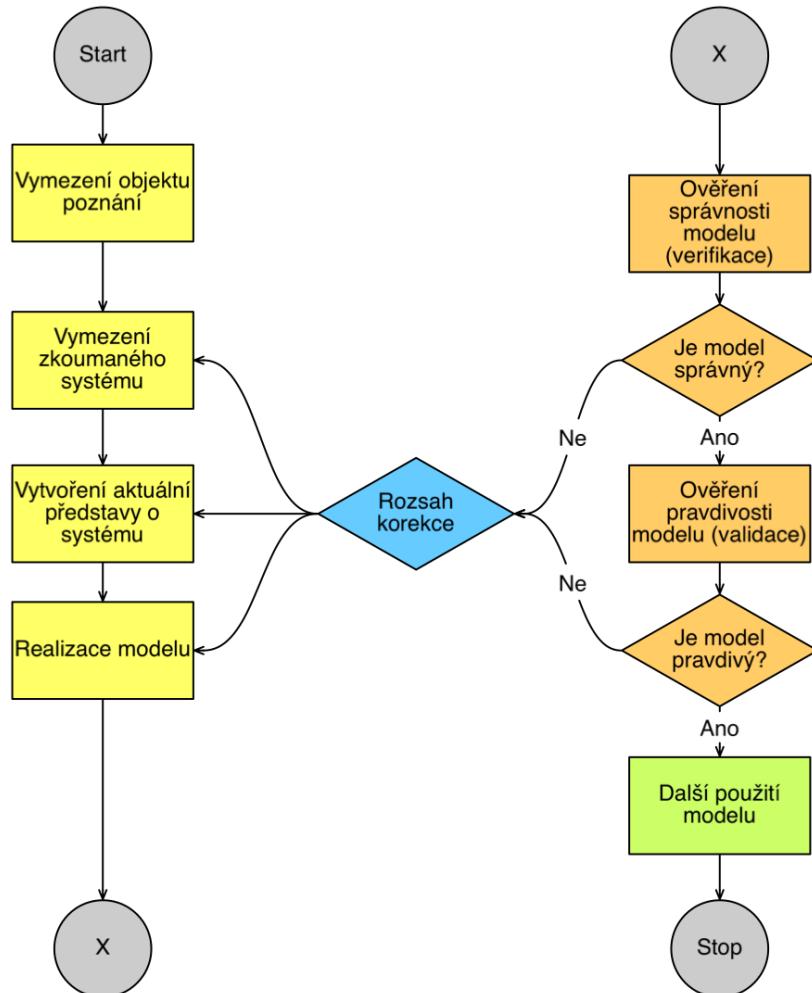
Máme-li **definovat předmět poznání (simulovaný systém)** jednoznačně, musíme určit především hledisko zkoumání daného objektu a zvolit odpovídající rozlišovací úroveň. Hledisko nazírání je dáné v první řadě účelem zkoumání daného objektu. V průběhu zkoumání objektu se ovšem rozlišovací úroveň (rozlišení) může s postupujícím poznáním měnit: zpravidla se zvyšuje, ale občas i snižuje, pokud poznáme, že je něco nadbytečné.

Aktuální představa o simulovaném systému v sobě zahrnuje aktuální znalosti o zkoumaném systému, jeho struktuře a časových změnách, resp. zpracování projektu systému a identifikaci použitých subsystémů.

V etapě **realizace modelu** jde o návrh simulujícího systému a jeho realizaci na vhodném simulátoru (nejčastěji na číslicovém počítači). Návrh modelu může, ale nemusí vycházet z matematického popisu aktuální představy o simulovaném systému. Za simulační se považuje jen takový model, jenž při napodobování dynamiky simulovaného systému zachovává uspořádání posloupnosti časových změn. V matematickém popisu modelu se rozlišují:

- stavové proměnné (veličiny reprezentující stav systému v kterémkoliv okamžiku jeho existence);
- přenosové funkce (vztahy vyjadřující interakce mezi prvky systému, resp. mezi prvky systému a okolím);

- vynucující funkce (vstupní veličiny nebo faktory ovlivňující chování systému);
- parametry (konstantní veličiny charakterizující systém).



Obrázek 11 - Vývojový diagram procesu simulace systému



Model se považuje za **správný**, jestliže konkrétní implementace realizuje náš abstraktní návrh simulované věci. Kontrolujeme tedy, zda model dělá to, co jsme navrhli. Snažíme se o 100% shodu. Ověřením **pravdivosti modelu** rozumíme, zda náš návrh modelu reflektuje chování reálného systému. Hodnotíme užitečnost modelu, tedy jak model plní svůj účel. Snažíme se zde o maximální shodu s chováním reálné věci, nikoliv však 100%. Z fází ověřování správnosti, resp. pravdivosti, modelu se v případě neúspěchu vracíme k fázím již absolvovaným. Způsob a rozsah korekce závisí přitom zejména na charakteru a závažnosti zjištěných nesrovonalostí.

Ověřeného modelu lze v procesu poznání využívat např. k identifikaci parametrů modelu, k prognózování, vědecké predikci, optimalizaci, ale též ve výuce, výcviku apod.

Kontrolní otázky a úkoly:

1. Jaké jsou základní fáze procesu simulace systému?
2. V čem spatřujete rozdíl mezi správností modelu a pravdivostí modelu?
3. Vysvětlete vlastními slovy postup při tvorbě simulačního modelu, jeho ověřování a následné aplikaci.

Pojmy k zapamatování (uvádíme jen základní metody):

- kvalitativní teorie řešení diferenciálních rovnic
- bifurkace
- Thomova teorie katastrof
- GUHA
- metody matematické statistiky
 - regresní analýza
 - korelační analýza
 - analýza časových řad
 - shluková analýza
- Lindenmayerovy systémy
- buněčné automaty
- modely založené na agentech



Shrnutí:

Základními matematickými prostředky pro modelování a simulaci jsou diferenciální rovnice, diferenční rovnice, lineární algebra, klasické množiny a fuzzy množiny. Z matematických metod se používají nejčastěji kvalitativní teorie diferenciálních rovnic, metoda kompartmentových modelů (systémů), metoda automatizovaného generování hypotéz (GUHA), metody matematické statistiky a metody založené na teoriích formálních jazyků a automatů. Základní fáze procesu simulace systému jsou vymezeny vývojovým diagramem na (Obrázek 11).



Úkoly k zamýšlení:

2. Pokuste se vysvětlit, v čem spočívá obecně rozdíl mezi matematickými prostředky a matematickými metodami.?



5 Organizace simulačního modelu

V této kapitole se dozvíte:

- Zobrazení stavu simulovaného systému
- Zobrazení stavových změn a času
- Synchronizace výpočtu
- Vstupy a výstupy simulačních programů

Po jejím prostudování byste měli být schopni:

- získali představu o struktuře jádra simulačního programu a konstrukci jeho základních datových struktur,
- naučili se používat základních pojmu jako simulační jádro, stavové změny, události, simulární čas,
- pochopili princip základního cyklu simulačního programu.

Klíčová slova této kapitoly:

Základní pojmy, modelování, simulace, model.

Doba potřebná ke studiu: 3 hodiny



Průvodce studiem

Je zřejmé, že pro formální popis činnosti výchozího systému není prakticky možné navrhnut obecně platná pravidla. Určitě však má smysl pokusit se navrhnout jakousi univerzální metodiku pro algoritmizaci simulačního modelu (simulátoru) a jeho počítačové implementace. Taková univerzální metodika se přirozeně nemůže týkat těch částí algoritmizace, které jsou specifické pro konkrétní simulační úlohy, ale jen tzv. simulačního jádra, tj. té části, jež je prakticky společná všem simulačním programům. Proto se v této kapitole seznámíte pouze s takovými datovými strukturami simulačního jádra, které jsou společné pro simulační modely diskrétního i spojitého typu.

5.1 Hlavní úkoly při konstrukci simulačního jádra

Hlavní úkoly při konstrukci simulačního jádra jsou:



1. navrhnut strukturu dat pro reprezentaci stavů simulovaného systému;
2. navrhnut operátory nad touto datovou strukturou, které realizují změny stavu systému;
3. zobrazit čas modelu a jeho průběh;
4. zajistit synchronizaci stavových změn systému tak, aby tyto změny probíhaly v určitém pořadí a při určitých hodnotách času nebo v okamžicích, kdy je splněna určitá podmínka týkající se stavu či konfigurace modelu.

5.2 Zobrazení stavů simulovaného systému

Pro reprezentaci stavů výchozího systému se používají nejrůznější datové struktury v závislosti na typu zvoleného programovacího jazyka. V nejjednodušším případě odpovídají skalárním stavovým veličinám jednoduché proměnné a vektorům datová pole.

Složitější datové struktury se využívají v případech, kdy se v průběhu času mění struktura systému nebo počet jeho prvků (např. při simulaci systémů hromadné obsluhy). V těchto případech je výhodné použít takových programovacích jazyků, jež umožňují přidělovat paměť exemplářům datových struktur dynamicky až v průběhu výpočtu. Objektově orientované programovací jazyky dovolují uživateli definovat (deklarovat) prakticky libovolně složité datové struktury a tyto dále obohacovat (koncepte tříd a podtříd). Např. při simulaci systémů hromadné obsluhy je velmi účelné použít spojových seznamů, které reprezentují fronty požadavků před obslužnými linkami.

5.3 Zobrazení stavových změn

Každá změna stavu je obecně realizována nějakým operátorem, který pracuje nad příslušnou datovou strukturou. Takový operátor sestává z posloupnosti příkazů, jejíž forma (makro, procedura, podprogram) je dána použitým programovacím jazykem. V moderních (objektově orientovaných) jazycích se tyto operátory logicky spojují přímo s odpovídajícími datovými strukturami (např. metody v deklaraci tříd jazyku Simula). Místo termínu „operátor“ používají někteří autoři slova „událost“, ale tuto praxi nedoporučujeme, protože termín „událost“ se používá v oblasti simulace v jiném významu.

5.4 Zobrazení času

Zobrazení času představuje specifický problém simulace.

Simulární čas

Čas výchozího (simulovaného) systému se zobrazuje v modelu pomocí aritmetické proměnné (buď typu real nebo typu integer). Pro tuto veličinu se zavádí speciální název - **simulární čas**.

Nedoporučujeme používat v češtině dosti rozšířeného termínu „simulovaný čas“, ani termínů „modelový čas“, resp. „systémový čas“. Přitom musí být splněny určité podmínky:

- hodnota simulárního času nesmí v průběhu výpočtu klesat,
- děje v simulačním modelu závisejí na simulárním čase stejným způsobem jako jejich vzory ve výchozím systému na toku přirozeného času.

Hodnota simulárního času by neměla být uživateli přímo přístupná. V simulačních programovacích jazycích se zpravidla dodržuje následující zásada: uživatel může hodnotu simulárního času zjišťovat (např. prostřednictvím vhodné funkční procedury), nikoliv však měnit.

Otzázkou k zamýšlení:

Pokuste se zdůvodnit, proč by uživatel neměl mít přímou možnost měnit hodnotu simulárního času.



5.5 Synchronizace výpočtu

Stav dynamického systému v konkrétním časovém okamžiku je určen hodnotami jeho stavových proměnných. Má-li simulovaný systém n stupňů volnosti, bude jeho stav popisován hodnotami s_1, s_2, \dots, s_n stavových proměnných S_1, S_2, \dots, S_n . Chceme-li v simulačním modelu sledovat vývoj takového systému, musíme v paměťovém prostoru vyhradit místo pro zobrazení jednotlivých stavových proměnných a pomocí vhodných programových prostředků zajistit aktualizaci hodnot těchto proměnných.

V případě diskrétní simulace předpokládáme, že se stav systému během konečného časového intervalu mění jen v konečně mnoha okamžicích. To znamená, že na spojité časové ose existuje pouze konečně mnoho časových okamžiků, v nichž se něco děje, v nichž nastávají tzv. **události (events)**. **Událostí přitom rozumíme změnu, jež je elementární a okamžitá (s nulovou dobou trvání)**. Podrobnosti nalezne čtenář v následující kapitole věnované algoritmizaci diskrétních simulačních modelů.

Událost

U spojitéch dynamických systémů, které se obvykle popisují soustavou diferenciálních rovnic, se hodnoty stavových proměnných mění spojitě. Nicméně i v tomto případě se počítají hodnoty stavových proměnných v určitých diskrétních časových okamžicích daných velikostí kroku zvolené integrační metody. Tyto okamžiky pak mají pro spojitou simulaci stejný význam jako okamžiky, v nichž se realizují události v diskrétní simulaci.

V následujícím odstavci se pokusíme detailněji formulovat základní cyklus simulačního programu. Nenechte se při studiu odradit tím, že přitom používáme matematické symboliky. Pro přesné pochopení toho, co Vám chceme sdělit, je to opravdu nutné. Správné pochopení následujícího textu Vám určitě usnadní schéma na (Obrázek 12).



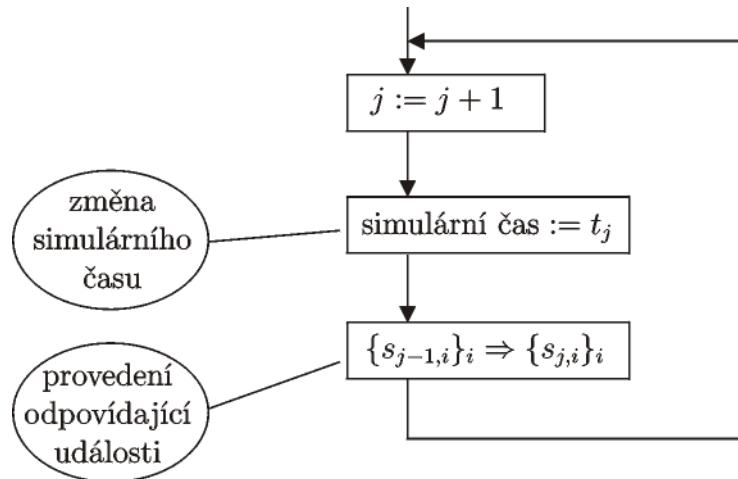
Uvažujme simulační model, v němž nastávají události při hodnotách simulárního času t_1, t_2, \dots, t_m . Chování modelu v průběhu časového intervalu $\langle T_z, T_k \rangle$, kde T_z značí začátek a T_k konec simulačního experimentu, pak udává konečná posloupnost

$$\{\{s_{j,i}\}_{i=1}^n, t_j\}_{j=1}^m$$

taková, že

1. $T_z \leq t_1, \quad t_m \leq T_k,$
2. $\forall j: 1 < j \leq m \Rightarrow t_{j-1} \leq t_j,$
3. posloupnost $\{s_{j,i}\}_{i=1}^n$ určuje stav modelu v simulárním čase t_j .

Změny stavu modelu se programově realizují pomocí posloupnosti příkazů, procedur nebo podprogramů. Požadované synchronizace mezi prováděním programové události a simulárním časem se dosáhne tak, že vždy po aktualizaci hodnoty simulárního času provedeme odpovídající programovou událost (viz schéma Obrázek 12).



Obrázek 12 - Schéma základního cyklu simulačního programu

5.6 Vstupy a výstupy simulačních programů

Je-li simulační model výchozího systému zkonztruován a jsou-li ověřeny jeho správnost a pravdivost, můžeme jej využít k vlastním simulačním experimentům. S organizací simulačních experimentů je spojena řada problémů. **Není-li chování výchozího systému deterministické, musí konstruovaný model přirozeně tuto skutečnost respektovat** (reflektovat), takže navrhovatel modelu musí specifikovat základní parametry stochastických procesů probíhajících ve výchozím systému.

Problémy související se stochastickým chováním modelovaných systémů lze podle rozdělit do tří skupin:

- specifikace parametrů rozdělení náhodných veličin,
- modelování procesů s náhodnými parametry,
- statistické hodnocení simulačních experimentů.

 Pokud navrhujeme simulační model dosud neexistujícího reálného systému, nezbývá nic jiného než odhadnout tyto parametry na základě zkušeností s chováním obdobných reálně existujících systémů.

V případě, kdy reálný systém, který modelujeme, již existuje, můžeme jeho chování sledovat po jistou dobu a potřebné parametry odhadnout na základě získaných experimentálních údajů. Experimentální data můžeme při specifikaci využít trojím způsobem:

- ke konstrukci teoretické distribuční funkce, tj. funkce dané exaktním vzorcem;
- ke konstrukci empirické distribuční funkce (zpravidla schodovité nebo po částech lineární), ježíž hodnoty se získají frekvenční analýzou experimentálních dat;
- přímo, pokud jsou získaná data dostatečně reprezentativní.

Pro experimentování se simulačním modelem se nejčastěji používá varianta první.

Pokud mluvíme o modelování procesů s náhodnými parametry, máme na mysli procesy, jejichž parametry mají charakter náhodných veličin.

Získáme-li distribuční funkci takových parametrů jistého procesu, pak jej můžeme v modelu napodobit, tj. vytvořit proces, jehož parametry budou mít rozdělení dané získanou distribuční funkcí. Pro generování hodnot náhodných veličin v modelu se používají tzv. generátory pseudonáhodných čísel. Této problematice je ve skriptech věnován samostatný odstavec 4.4.

S vytvořenými simulačními modely provádíme experimenty, jejichž výsledky podléhají statistickému zpracování. Přitom se požaduje, aby jednotlivé simulační experimenty byly statisticky nezávislé. Obecně platí, že **výsledky simulačních experimentů s odlišnou (nezávislou) inicializací generátorů pseudonáhodných čísel uspokojivě splňují požadavky na nezávislost**. Jak však zajistit nezávislost výsledků různých experimentů? V praxi se uplatňují čtyři odlišné přístupy:

1. Triviální je provedení série nezávislých simulačních experimentů, tj. série samostatných běhů simulačního programu s různou inicializací generátorů pseudonáhodných čísel.
2. V případě relativně stabilizovaného systému někdy postačí realizovat jediný simulační experiment a vybrat data, která jsou od sebe časově dostatečně vzdálena (tj. výsledky nezávislých úseků simulačního experimentu). Nezávislost vybraných úseků simulačního pokusu je nutno otestovat.
3. Při simulaci systémů (malé nároky na paměť počítače) je možno postupovat tak, že se v simulačním programu modeluje hypotetický systém sestávající z více kopií výchozího systému, které pracují vzájemně nezávisle, samozřejmě podle stejných pravidel.
4. Speciální přístup je použitelný u tzv. **regenerativních systémů**, pro něž existuje taková rostoucí posloupnost časových okamžiků (tzv. **regenerativních okamžiků**) $\{b_i\}$, že chování systému mezi libovolnými dvěma po sobě následujícími okamžiky b_i a b_{i+1} je nezávislé na historii a všechny parametry ovlivňující chování systému v těchto intervalech mají identické rozdělení. V takovém případě lze chování systému v jednotlivých intervalech $\langle b_i, b_{i+1} \rangle$ považovat za nezávislé.

Pojmy k zapamatování:

- simulační jádro
- událost
- simulární čas
- nezávislost simulačních experimentů

Kontrolní otázky:

1. Jaké struktury se používají pro reprezentaci stavů simulovaného systému?
2. Co je to simulární čas?
3. Jakým způsobem zajistíme synchronizaci výpočtu?
4. Jak nastavujeme parametry simulačních programů?





Shrnutí obsahu kapitoly

Při konstrukci simulačního jádra je nutno navrhnout vhodnou strukturu dat pro popis stavů simulovaného systému, navrhnout algoritmy realizující změny stavu, zobrazit čas modelu a synchronizovat stavové změny v průběhu času. Základní cyklus simulačního programu je schematicky znázorněn na obr. 3.1. Jestliže je chování simulovaného systému stochastické (náhodné), musí tuto skutečnost respektovat i příslušný simulační model. Při statistickém vyhodnocování experimentů prováděných se simulačními modely se požaduje, aby jednotlivé experimenty byly statisticky nezávislé. Splnění tohoto požadavku se nejsnáze dosáhne tím, že se realizuje série experimentů s různými násadami generátorů pseudonáhodných čísel.

6 Základy teorie diskrétních dynamických systémů

V této kapitole se dozvítíte:

- Definice diskrétního dynamického systému
- Stacionární trajektorie
- Analýza diskrétního dyn. systému

Po jejím prostudování byste měli být schopni:

- pochopíte základy teorie diskrétních dynamických systémů s jedinou stavovou proměnnou,
- naučíte se používat základní pojmy jako diskrétní dynamický systém, trajektorie bodu, pevný bod, periodické body, cyklus,
- naučíte se určovat stacionární trajektorie takových systémů a posuzovat jejich stabilitu.

Klíčová slova této kapitoly:

Diskrétní dynamický systém, trajektorie.

Doba potřebná ke studiu: 4 hodiny

Průvodce studiem

V této kapitol se stručně seznámíte se základy teorie diskrétních dynamických systémů. Tato teorie je propracována především pro případ jediné stavové proměnné, a proto se i naše další úvahy omezí na tento případ. Pro popis chování diskrétního dynamického systému jsou využívány diferenční, resp. rekurentní rovnice. Tyto rovnice udávají, jak se mění hodnoty stavových proměnných.



6.1 Definice diskrétního dynamického systému



Diskrétní dynamický systém je matematická struktura určená třemi složkami:

1. intervalom J , v němž leží všechny možné hodnoty stavové proměnné x , např.
 $J = \langle a, b \rangle$, kde $a < b$;
2. funkcí f , definovanou na intervalu J ;
3. diferenční rovnicí

$$x_{n+1} = f(x_n),$$

jenž reprezentuje "pohybovou" rovnici uvažovaného systému.

Vyjdeme-li (v čase $t = 0$) z nějaké hodnoty x_0 , potom posloupnost $x_0, x_1 = f(x_0), x_2 = f(x_1), \dots$ nazýváme **trajektorie** bodu x_0 , což není nic jiného než řešení dynamického systému s počátečním stavem x_0 .

Trajektorie bodu

Pro libovolnou funkci f a libovolné celé kladné n můžeme zavést složené funkce f^n tímto předpisem

$$f^1(x) = x, f^2(x) = f(f(x)), \dots f^{n+1}(x) = f(f^n(x)).$$

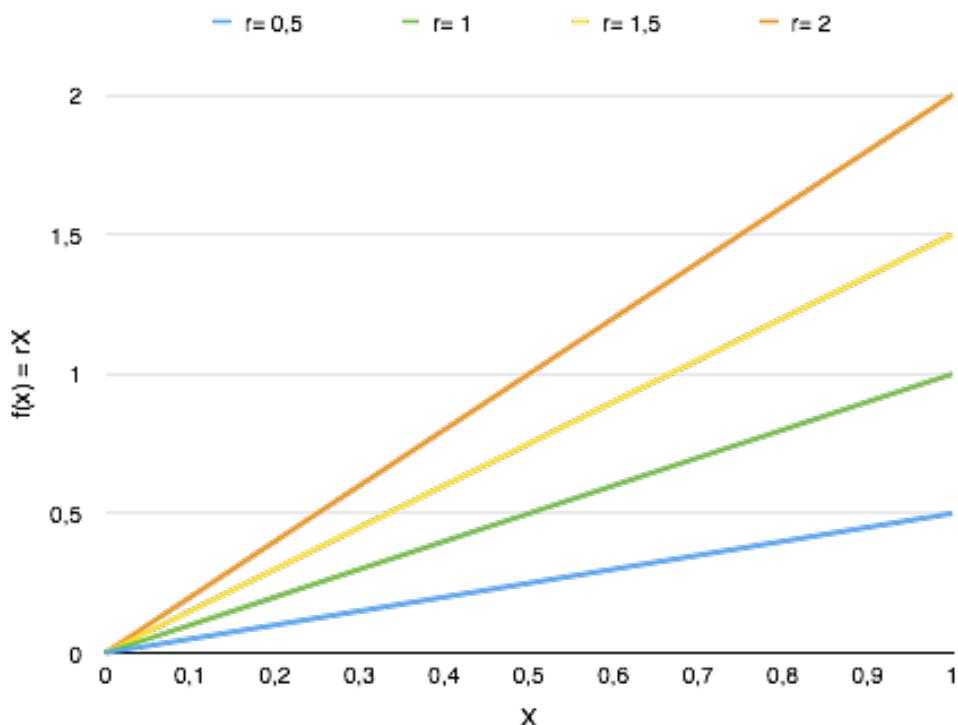
Takto definovaná funkce $f^n(x)$ se nazývá **n-tá iterace funkce f** .



Příklad. Mějme následující pohybovou rovnici uvažovaného systému.

$$X_{n+1} = f(rX_n) \quad (6.1)$$

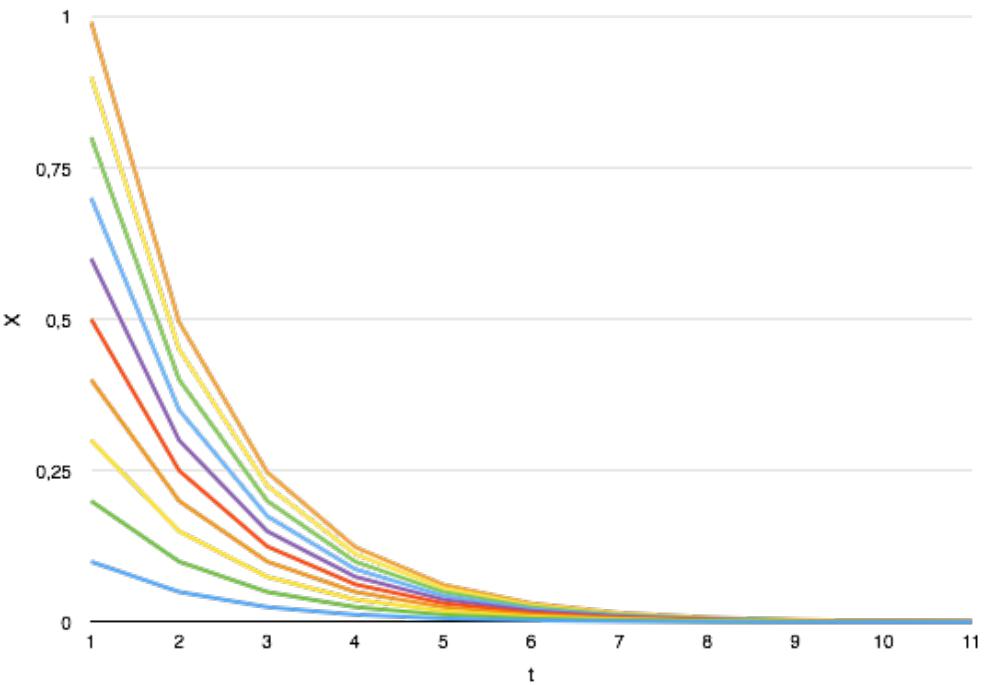
Jde o jednoduchý model pro růst populace. Velikost populace značíme X , parametr $r = p - u$ označuje míru reprodukce a je roven rozdílu míry porodnosti a úmrtnosti. Parametr r značí směrnici přímky v grafu (Obrázek 13). Interval J je v tomto případě $J = \langle 0,1 \rangle$.



Obrázek 13 – Pohybová rovnice modelu pro růst populace

Pro různou hodnotu parametru r dostaneme přímkou s danou směrnicí odpovídající právě parametru r .

Trajektorii získáme dosazením do rovnice (6.1, Obrázek 13) pro zvolený hodnotu parametru r . Na (Obrázek 14) vidíme trajektorie bodu pro různé počáteční hodnoty x_0 a hodnotu parametru $r = 0,5$. Na ose x se nachází krok výpočtu, který může znamenat určitý časový úsek. Na ose y je poté hodnota populace v daném kroku. V tomto případě dojde k vyhynutí populace pro jakoukoliv výchozí hodnotu populace x_0 .



Obrázek 14 – Trajektorie bodu x_0 pro $r = 0,5$ rovnice (6.1)

Podobným způsobem lze vypočítat vývoj populace popsané danou rovnicí (6.1) pro zvolenou hodnotu parametru r a výchozí hodnotu této populace.

6.2 Stacionární trajektorie diskrétního dynamického systému

Mezi trajektoriemi daného dynamického systému zaujímají význačné postavení tzv. **stacionární trajektorie**, které odpovídají ustálenému pohybu systému. Jsou to:

- pevné body,
- periodické trajektorie (cykly).

Bod $\alpha \in J$ je **pevným bodem** funkce f , jestliže platí $f(\alpha)=\alpha$. Trajektorie pevného bodu α je zřejmě stacionární, všechny členy posloupnosti jsou totiž stejné (rovné α).

Pevné body funkce f se určí jednoduše: jsou to právě všechna řešení rovnice $f(x) = x$. Existují přirozeně různé typy pevných bodů.

Pevný bod α funkce f je:

- **atrahující (přitahující)**, jestliže existuje takový otevřený interval V obsahující α , že se trajektorie libovolného bodu $x_0 \in V$ (posloupnost $x_0, f(x_0), f^2(x_0), \dots$) blíží postupně k α ;
- **repulzivní (odpuzující)**, jestliže existuje takový otevřený interval V obsahující α , že trajektorie libovolného bodu $x_0 \in V$ ($x_0 \neq \alpha$) "vyběhne" po nějakém čase z intervalu V , tj. pro nějaké n platí $f^n(x_0) \notin V$.

Stacionární trajektorie

Pevný bod

Existují ovšem i takové pevné body, které nejsou ani atrahující, ani repulzivní.

Následující věta umožňuje rozhodnout, zda uvažovaný pevný bod α funkce f je atrahující či repulzivní.

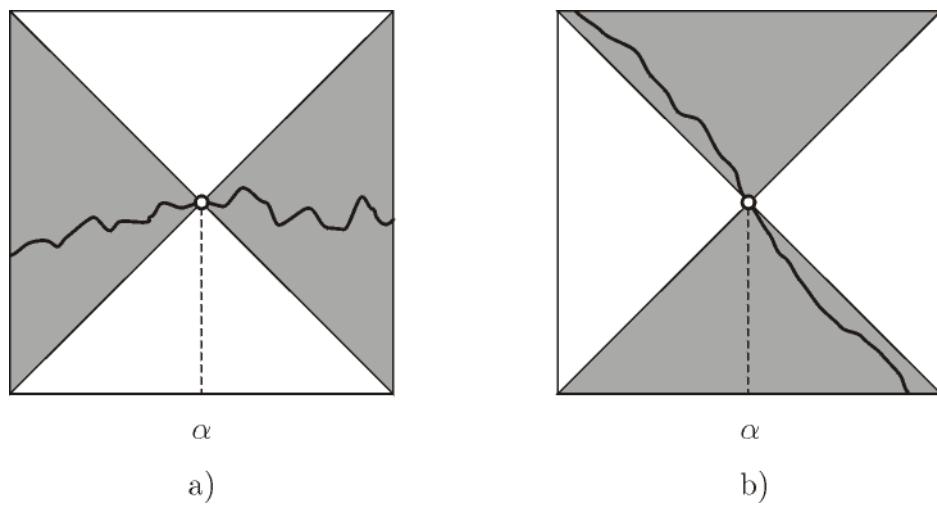
Věta 1. Jestliže pro libovolné $x \in V, x \neq \alpha$ platí:

$$1. \left| \frac{f(x) - f(\alpha)}{x - \alpha} \right| < 1, \quad (6.2)$$

pak α je atrahující pevný bod;

$$2. \left| \frac{f(x) - f(\alpha)}{x - \alpha} \right| > 1, \quad (6.3)$$

pak α je repulzivní pevný bod.



Obrázek 15 - Ilustrace k určení typu pevného bodu

Podmínky (6.2) a (6.3) věty 1 je možno snadno interpretovat graficky (Obrázek 15). Uvažovaný pevný bod α je atrahující, jestliže graf funkce f (v jistém okolí bodu α) leží ve vyšrafovane oblasti na obr. 4.1a, a repulzivní, pokud leží ve vyšrafovane oblasti na (Obrázek 15b).

Druhým případem stacionárního chování systému jsou periodické trajektorie, tvořené posloupnostmi, v nichž se jistý počet členů (tzv. cyklus) "do nekonečna" opakuje. Trajektorie tohoto typu sestávají z periodických bodů.

Bod $\alpha_0 \in J$ je **periodickým bodem řádu n funkce f**, jestliže platí $f^n(\alpha_0) = \alpha_0$, přičemž $f^j(\alpha_0) \neq \alpha_0$ pro všechna $j = 1, 2, \dots, n-1$. Trajektorie takového bodu má tvar

$$\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_{n-1}, \alpha_0, \alpha_1, \alpha_2, \dots, \alpha_{n-1}, \alpha_0, \dots,$$

je to tedy periodická posloupnost s periodou n . Body $\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_{n-1}$ tvoří **cyklus řádu n** . (Pevný bod je podle této definice periodickým bodem řádu právě 1.)

Jak hledat periodické body řádu n pro danou funkci f ? Také tato úloha je principiálně jednoduchá. V prvním kroku musíme nalézt všechna řešení rovnice

$$f^n(x) = x.$$

Z těchto řešení pak (ve druhém kroku) vyloučíme ta, jež představují pevné body funkce f a periodické body nižších řádů m (m je dělitelem n).

V praxi se často setkáváme s trajektoriemi, které sice nejsou přesně periodické (jako trajektorie periodických bodů), ale s postupem času se k nim přibližují. Takové trajektorie se nazývají **asymptoticky periodické**.

Z uvedeného je zřejmé, že má smysl mluvit o atrahujících a repulzivních cyklech.

Cyklus $\alpha_0, \alpha_1, \dots, \alpha_{k-1}$ řádu k funkce f je

- **atrahující**, když alespoň jeden bod tohoto cyklu je atrahujícím pevným bodem funkce f^k ;
- **repulzivní**, když každý bod tohoto cyklu je repulzivním pevným bodem funkce f^k .

Cyklus

Právě uvedená definice zřejmě dovoluje rozhodnout o typu cyklu (jeho atrahovatelnosti či repulzivnosti) postupem, který jsme již vysvětlili v případě pevných bodů.

6.3 Analýza konkrétního diskrétního dynamického systému

Uvažujme dynamický systém určený funkcí

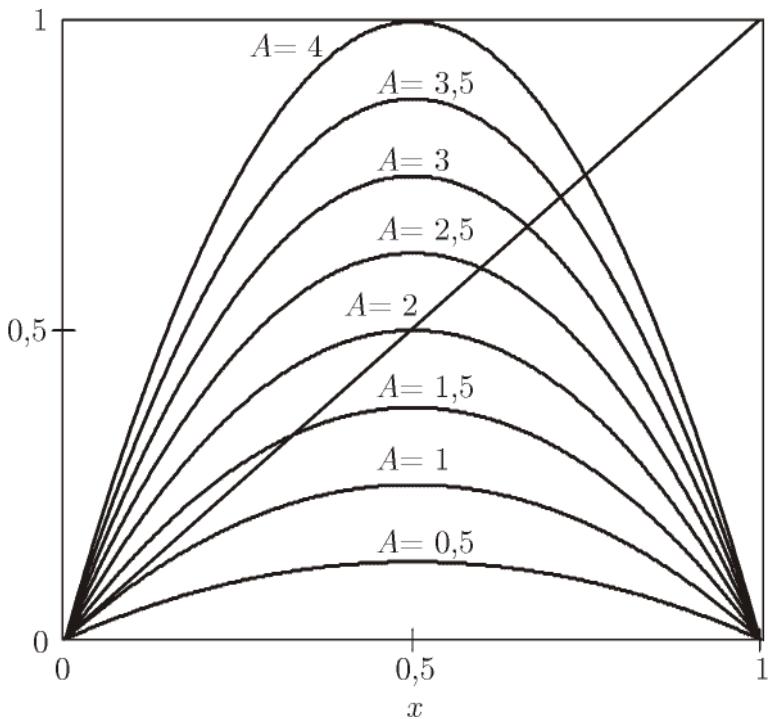
$$f(x) = Ax(1 - x), \quad (6.4)$$



v níž stavová proměnná x reprezentuje relativní četnost nějaké populace v prostředí s omezenými zdroji ($x \in \langle 0, 1 \rangle$) a A je reálný parametr charakterizující rychlosť růstu uvažované populace ($A > 0$).

Pevné body funkce (6.4) získáme jednoduše řešením rovnice

$$Ax(1-x) = x. \quad (6.5)$$

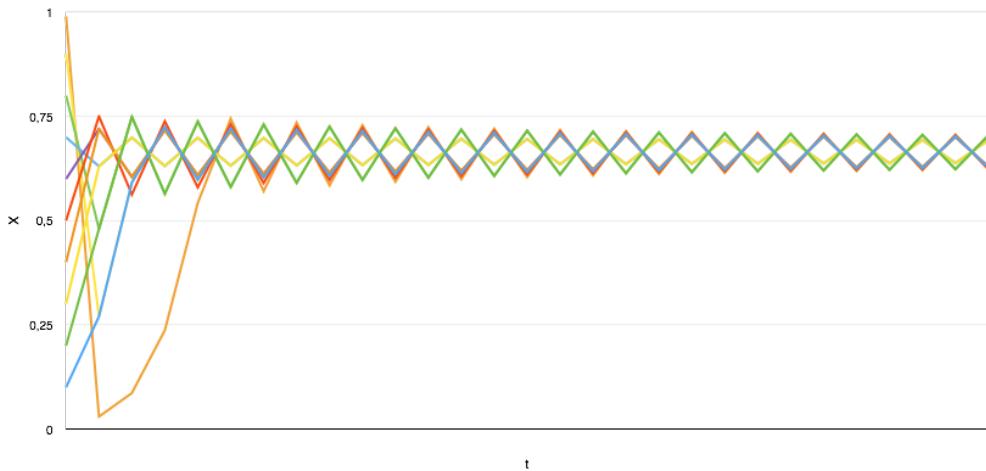


Obrázek 16 - Ilustrace k pevným bodům funkce $Ax(1-x)$

Pro přehlednost rozlišíme tři případy.

- 1 V případě $A \in \langle 0,1 \rangle$ má funkce (6.4) právě jediný pevný bod $\alpha = 0$ (Obrázek 16). Její graf leží celý pod grafem přímky $y = x$, takže uvedený pevný bod je atrahující. To ovšem znamená, že se trajektorie libovolného bodu $x_0 \in \langle 0,1 \rangle$ neomezeně blíží k nule. Biologická interpretace je jednoduchá: populace s tak malou rychlostí růstu zákonitě vymírá.
- 2 Jak je patrno z (Obrázek 16), pro $A \in (1,3)$ existují právě dva pevné body funkce (6.4), a to $\alpha = 0$ a $\beta = 1 - 1/A$. Původně jediný pevný bod $\alpha = 0$ se "rozštěpí" na dva, přičemž nový pevný bod $\beta = 1-1/A$ se s rostoucí hodnotou parametru A vzdaluje od počátku. Můžeme snadno ukázat, že $\alpha = 0$ je nyní repulzivní pevný bod (graf funkce (6.4) v jeho okolí leží nad grafem $y = x$), kdežto $\beta = 1-1/A$ je atrahující pevný bod. Tuto skutečnost si můžeme ověřit numerickým výpočtem. Trajektorie libovolného bodu x_0 se v tomto případě tedy přiblížuje neomezeně k hodnotě $1-1/A$, což znamená, že relativní četnost populace postupně roste k uvedenému maximu (stavu nasycení).
- 3 V případě $A \in (3,4)$ je situace velmi složitá. Při tak vysokých hodnotách růstového parametru A se mohou vyskytnout jak periodické trajektorie, tak i trajektorie asymptoticky periodické nebo dokonce zcela nepravidelné (chaotické).

Na (Obrázek 17) můžeme vidět trajektorii bodu x_0 pro parametr $A = 3$. Hodnota populace osciluje mezi dvěma body, ale postupně se přiblížuje neomezeně k jedné hodnotě.



Obrázek 17 - Trajektorie bodu x_0 pro $A = 3$ rovnice (6.4)

Ukážeme, jak nalézt periodické body řádu 2 funkce (6.4), tj. trajektorii $\gamma_1, \gamma_2, \gamma_1, \gamma_2, \dots$, kde $f(\gamma_1) = \gamma_2$ a $f(\gamma_2) = \gamma_1$. Zmíněné body najdeme řešením rovnice $f^2(x) = x$, tj.

$$A[Ax(1-x)][1-Ax(1-x)] = x.$$

Tuto rovnici lze přepsat ve tvaru

$$x[x - (1 - 1/A)][A^2x^2 - (A^2 + A)x + (A + 1)] = 0, \quad (6.6)$$

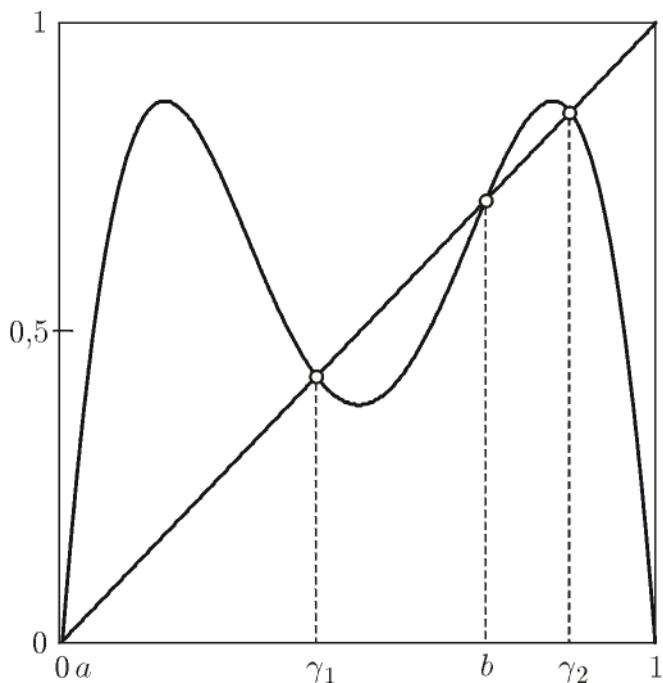
z něhož je zřejmé, že řešením (6.6) jsou oba již zmíněné pevné body $a = 0$ a $b = 1 - 1/A$. Vyloučíme-li tato řešení, dostaneme obyčejnou kvadratickou rovnici

$$A^2x^2 - (A^2 + A)x + (A + 1) = 0. \quad (6.7)$$

Její diskriminant

$$D = A^4 - 2A^3 - 3A^2 = A^2(A + 1)(A - 3)$$

je pro $A > 3$ zřejmě kladný, takže rovnice (6.7) má právě dva kořeny γ_1 a γ_2 s požadovanou vlastností. Numerickým řešením např. pro $A = 3,5$ dostaneme $\gamma_1 = 0,42857$ a $\gamma_2 = 0,85714$.



Obrázek 18 - Ilustrace k určení periodických bodů řádu 2 funkce $Ax(1-x)$

Existenci těchto dvou periodických bodů γ_1 a γ_2 si můžeme názorně ověřit na (Obrázek 18). Relativní četnost populace v takovém případě periodicky kolísá mezi hodnotami γ_1 a γ_2 .



Úkol k textu:

Analyzujte chování systému popsaného funkcí $Ax(1-x)$ pro hodnoty parametru A rovné 0,5, 2 a 4.



Pojmy k zapamatování:

- diskrétní dynamický systém
- trajektorie bodu
- pevný bod a jeho stabilita
- periodické body
- cyklus



Shrnutí:

V této kapitole jste se především seznámili s definicí diskrétního dynamického systému. Tato definice je podle našeho názoru natolik „průhledná“, že Vám její pochopení nebude činit potíže. Dále jste se naučili, jak postupovat při určování stacionárních trajektorií (pevných a periodických bodů) jednoduchých diskrétních dynamických systémů a také jak posuzovat jejich stabilitu (např. „graficky“).

7 Algoritmizace diskrétních simulačních modelů

V této kapitole se dozvíte:

- Procesy a jejich vnější stavy
- Přechody mezi vnějšími stavami procesů
- Vnitřní stavu procesů
- Kalendáře událostí a jeho implementace
- Plánování událostí

Po jejím prostudování byste měli být schopni:

- pochopit strukturu simulačního jádra diskrétních simulačních modelů,
- naučit se používat základní pojmy jako událost, proces, kalendář událostí, plánovací příkaz.

Klíčová slova této kapitoly:

Procesy, stavy procesů, kalendář událostí, plánování událostí.

Doba potřebná ke studiu: 4 hodiny

Průvodce studiem

V této kapitole se podrobně zabýváme problematikou výstavby diskrétního simulačního modelu s důrazem na konstrukci simulačního jádra programu (události, procesy, kalendář událostí, plánování procesů). Uvědomte si, že právě naprogramování simulačního jádra je základním úkolem diskrétní simulace, a proto věnujte této kapitole zvýšenou pozornost.



Diskrétní simulační modely jsou charakterizovány tím, že všechny stavové proměnné nabývají pouze diskrétních hodnot a v průběhu času se mění skokem. Nejčastějším případem diskrétních modelů jsou aplikace teorie hromadné obsluhy.

Pro diskrétní simulační modely jsou charakteristické následující rysy:

- proměnný počet prvků systému (požadavků),
- reprezentace front pomocí spojových seznamů,
- vysoký stupeň paralelnosti výpočtu,
- velké nároky na řízení programu (vyplývá z paralelnosti),
- vysoké nároky na paměť (velký počet prvků - požadavků).

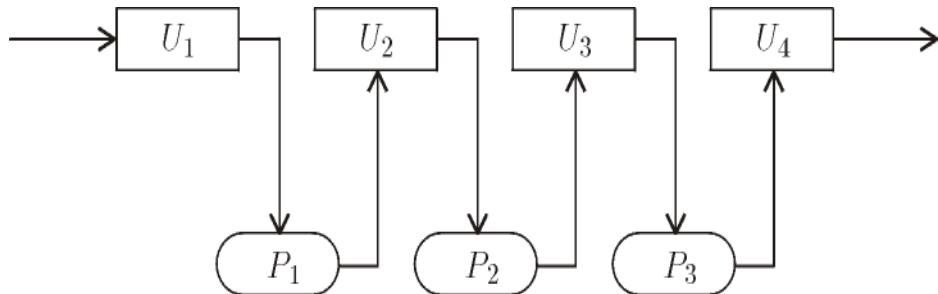
7.1 Procesy

Na rozdíl od toho, co nabízí teorie, jsou diskrétní systémy, s nimiž se setkáváme v průmyslu, společnosti a komunikaci, obvykle vícerozměrné, dokonce s rozměrem, který se dynamicky mění v čase nebo je předem nepredikovatelný. V simulaci takových systémů hraje klíčovou roli pojem procesu.





Při diskrétní simulaci existuje na spojité časové ose jen konečný počet okamžiků, v nichž nastávají změny stavových veličin (události), které chápeme jako elementární a okamžité. Nositeli událostí jsou přirozeně objekty, přitom každá konkrétní posloupnost událostí je výsledkem složité interakce objektů. Většina událostí je takové povahy, že je lze zcela přirozeně vázat do složitějších celků, tzv. procesů (simulačních procesů). **Proces (process)** je tedy posloupnost logicky na sebe navazujících událostí. Strukturu procesu můžeme obecně popsat schématem uvedeným na obr. 5.1.



Obrázek 5.1: Obecné schéma procesu

Proces se tedy neprovádí celý najednou (v jediném časovém okamžiku). V jednom určitém časovém okamžiku se tedy realizuje pouze část procesu odpovídající právě jediné události. Jednotlivé části procesu (události) U_i jsou od sebe odděleny tzv. plánovacími příkazy (příkazy potlačení procesu) P_i , jež způsobí, že se daný proces přestane provádět a začne se realizovat proces jiný. Při dalším vyvolání (aktivaci) se uvažovaný proces nezačíná provádět od začátku, ale od místa, kde byl naposledy přerušen.

Je-li tedy nějaký proces aktivován v jistém časovém okamžiku, provede se při této hodnotě simulárního času jen část uvažovaného procesu, a to právě událost, která se ve schématu na obr. 5.1 nachází bezprostředně za posledním realizovaným potlačením procesu. S potlačením procesu je přirozeně spojeno předání řízení výpočtu obecně jinému procesu. To umožňuje modelovat simultánně probíhající procesy pomocí sekvenčně prováděných instrukcí na jednoprocесорovém počítači.

7.2 Vnější stavy procesů

V souvislosti s plánováním procesů se rozlišují **čtyři vnější stavy procesů**:

Stav aktivní

1. **Stav aktivní (active).** Proces je v aktivním stavu, je-li právě prováděn, tj. je-li právě realizován výpočet odpovídající některé jeho události. V aktivním stavu může být v daném okamžiku výpočtu nejvíše jeden proces.

Stav ukončený

2. **Stav ukončený (terminated).** Proces se nachází ve stavu ukončeném, je-li je ukončena jeho operační část. Jestliže tato operační část obsahuje příkazy skoků, pak nemusí být zcela vyčerpána posloupnost událostí, z nichž se uvažovaný skládá. Takový proces již nemůže být ani aktivován, ani naplánován k provádění.

3. **Stav připravený neboli suspendovaný (suspended).** Proces v suspendovaném stavu není sice aktivní, ale je naplánován k provádění v nějakém konkrétním okamžiku simulárního času. Pokud nebude jeho naplánování zrušeno jinými procesy nebo simulační pokus neskončí, dojde k jeho vyvolání.
4. **Stav pasivní (passive).** V pasivním stavu je takový proces, který není ukončen, ale není také momentálně naplánován k provedení. K jeho vyvolání může dojít tehdy, bude-li naplánován prostřednictvím nějakého jiného procesu.

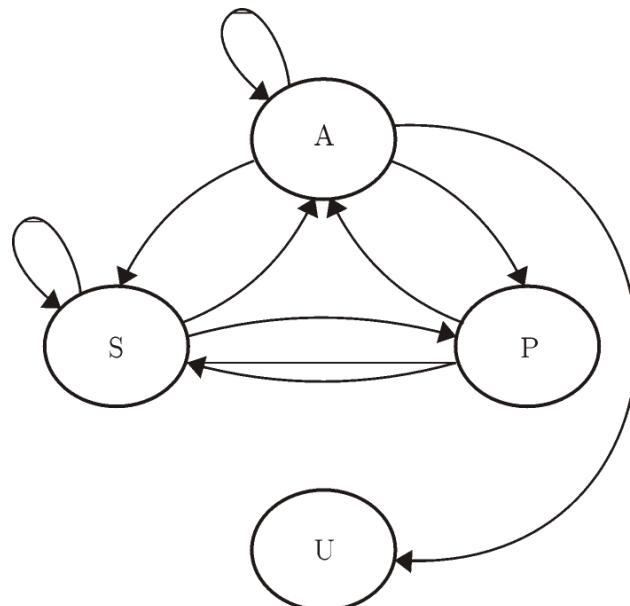
Stav připravený

Stav pasivní

Jakýkoliv proces vytvořený v průběhu výpočtu simulačního programu se v každém okamžiku vyskytuje právě v jednom z uvedených stavů.

7.3 Přechody mezi vnějšími stavy procesů

Na (Obrázek 19) jsou schematicky znázorněny všechny možné **přechody mezi vnějšími stavy procesů**.



Obrázek 19 - Schéma možných přechodů mezi vnějšími stavy procesů

Z toho, co bylo už dříve řečeno, je zřejmé, že přechody ze stavu ukončeného do jiných stavů jsou principiálně nemožné. Přechod procesu do ukončeného stavu se děje implicitně, jsou-li ukončeny výpočty v jeho operační části.

V této části se budeme podrobněji zabývat základními typy změn vnějšího stavu procesu.

Změna stavu aktivní → suspendovaný. K této změně dochází tehdy, je-li potlačeno provádění právě aktivního procesu a pokračování tohoto procesu je naplánováno až po uplynutí jisté doby. V popsané situaci je možné, že nějaký jiný proces je naplánován k provedení v simulárním čase menším, než je čas pokračování dosud aktivního procesu, nebo ve stejném simulárním čase, ale s vyšší prioritou. V tomto případě se aktivním stane

právě takový proces, který je naplánován k vyvolání nejdříve (proces s minimální hodnotou simulárního času, popř. proces, jenž je naplánován k vyvolání ve stejném čase jako pokračování původně aktivního procesu a má přitom nejvyšší prioritu).

Ke změně stavu aktivní → suspendovaný může dojít i tehdy, když právě aktivní proces přímo vyvolá provádění procesu jiného.

Změna stavu aktivní → aktivní. V právě popsaném případě se ovšem může stát, že žádný jiný proces není naplánován s hodnotou simulárního času menší, než je hodnota, při níž má právě aktivní proces pokračovat, nebo se stejnou hodnotou simulárního času, ale vyšší priorotou než právě aktivní proces. Pak dojde pouze ke změně hodnoty simulárního času a dosud aktivní proces zůstává v aktivním stavu i nadále.

Změna stavu aktivní → pasivní. V podstatě jde o potlačení právě aktivního procesu, přičemž jeho pokračování není ještě explicitně naplánováno. Dosud aktivní proces setrvává v pasivním stavu, dokud jej nějaký jiný proces nepřevede do stavu suspendovaného nebo přímo aktivního. Podobně jako v předcházejících případech se po potlačení dosud aktivního procesu převede do aktivního stavu ten proces, jenž je naplánován k provedení nejdříve.

Změna stavu aktivní → ukončený. K této změně dojde při vyčerpání operační části uvažovaného procesu. Přitom se ukončí provádění tohoto procesu a řízení výpočtu se předá procesu, který je naplánován k provedení nejdříve. Při ukončení operační části nějakého význačného procesu může dojít k ukončení celého simulačního experimentu.

Změna stavu suspendovaný → aktivní. Taková změna může nastat dvojím způsobem:

- nepřímo při potlačení jiného procesu, který byl dosud aktivní;
- přímo tím, že zrušíme existující naplánování nějakého procesu a naplánujeme jej znova s časem rovným momentální hodnotě simulárního času. Přitom nově aktivní proces se provádí při stejné hodnotě simulárního času jako dosud aktivní proces. Dosud aktivní proces je až druhý v pořadí, právě za nově aktivovaným procesem.

Změna stavu suspendovaný → pasivní. K této změně dojde jednoduše při zrušení již existujícího plánu uvažovaného procesu.

Změna stavu suspendovaný → suspendovaný. V tomto případě proces zůstává i nadále v suspendovaném stavu. Změní se pouze hodnota simulárního času, v němž je naplánována aktivace uvažovaného procesu.

Změna stavu pasivní → suspendovaný. Tato změna nastane, když naplánujeme budoucí aktivaci momentálně nenaplánovaného procesu.

Důkladně si promyslete jednotlivé případy přechodů mezi vnějšími stavami procesů. Usnadní Vám to pochopení dalšího výkladu týkajícího se kalendáře událostí.



7.4 Vnitřní stavы procesů

Poměrně složitý systém řízení simulačních výpočtů (potlačování právě aktivních procesů a aktivace jiných s minimálním časovým plánem aktivace) umožnuje na druhé straně zjednodušit jiné části simulačního programu. To se týká především zobrazování tzv. **vnitřních stavů jednotlivých procesů**. Významnou, nikoli však jedinou, charakteristikou vnitřního stavu procesu je jeho reaktivitační bod. **Reaktivitačním bodem** procesu přitom rozumíme právě to místo v operační části procesu, v němž je výpočet uvažovaného exempláře procesu přerušen a od něhož bude výpočet pokračovat při následující aktivaci tohoto exempláře.

Reaktivitační bod

7.5 Kalendáře událostí a jejich základní funkce

Kalendář událostí nebo také **simulační kalendář** (např. v jazyku SIMULA **sequencing set**) je řídící struktura simulačního programu, která zahrnuje především programové prostředky pro plánování událostí. Posloupnost událostí v simulačním modelu není přirozeně dána předem. Proto je základním úkolem simulačního programu tuto posloupnost vytvářet a průběžně aktualizovat. A je to právě kalendář událostí, jenž musí plnění tohoto úkolu zajišťovat.



Simulační programovací jazyky zpravidla již obsahují standardní prostředky pro plánování událostí. Pokud však implementujeme simulační model v nějakém jazyku, který nemá k dispozici standardní prostředky pro plánování událostí, musíme celé simulační jádro programu včetně plánovacích prostředků vytvořit sami.

Vyžaduje se, aby kalendář událostí zabezpečoval čtyři **základní funkce** (Malík 1989):

1. zjistit, zda je daná událost (elementární část nějakého procesu) naplánována či nikoliv, a v případě, že naplánována skutečně je, zjistit hodnotu jejího aktivačního času;
2. vybrat proces s minimální hodnotou aktivačního času a pokud je takových procesů se stejnou hodnotou aktivačního času více, vybrat ten, který má nejvyšší prioritu;
3. naplánovat momentálně nenaplánovanou událost (proces);
4. zrušit plán momentálně naplánované události (procesu).

Základní funkce

Kalendář událostí s uvedenými funkcemi zajišťuje tzv. **časové plánování událostí**.

Časové plánování událostí

Datová struktura kalendáře událostí může mít samozřejmě celou řadu vzájemně odlišných implementací. Jednotlivé **implementace** se liší především výpočetní složitostí a efektivitou provádění základních funkcí. Výběr implementace závisí přirozeně na charakteru řešené simulační úlohy.

Implementace kalendáře událostí

Požadavkům na kalendář událostí vyhovuje libovolná struktura, která dovoluje:

- lineární uspořádání množiny právě všech událostí simulačního programu podle jejich aktivačních časů a stanovených priorit (existuje-li více událostí se stejným aktivačním časem),
- prohledávání této množiny podle číselného klíče (aktivačního času události) s přihlédnutím ke stanoveným prioritám.



Kalendář událostí je v podstatě uspořádaný seznam, jehož prvky obsahují informace o aktivačním čase dané události a její příslušnosti nějakému procesu.

7.6 Implementace kalendáře událostí

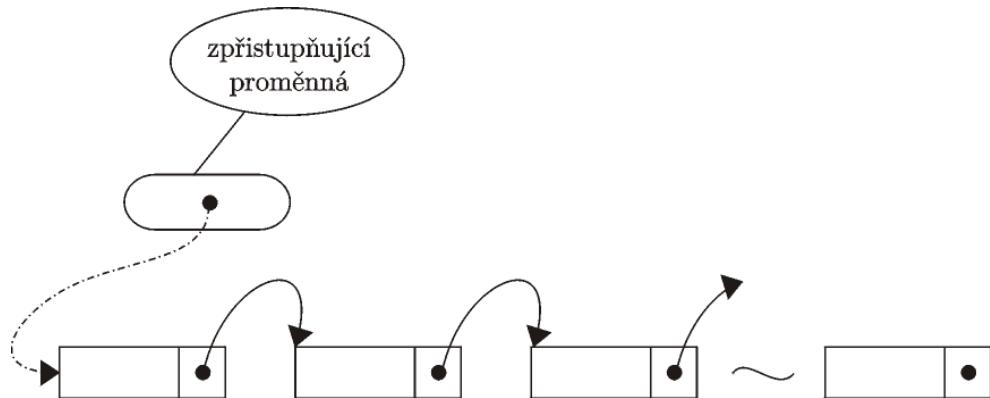
Nejjednodušší implementací simulačního kalendáře jsou uspořádané lineární seznamy (lineární spojové seznamy). Právě takové implementace se využívá u většiny univerzálních, pro simulaci vhodných, jazyků.

Lineární spojové seznamy

V případě **lineárních spojových seznamů** rozlišujeme:

- jednosměrné spojové seznamy,
- dvousměrné spojové seznamy.

Jednosměrné spojové seznamy jsou zpravidla zpřístupněny jedinou referenční proměnnou, která ukazuje na první prvek seznamu. Každý prvek pak odkazuje na svého následníka (Obrázek 20). Takové seznamy jsou vhodné ke konstrukci zásobníku pracujícího v režimu LIFO. Pro modelování fronty pracující v režimu FIFO lze také použít jednosměrného spojového seznamu, ovšem zpřístupněného dvěma referenčními proměnnými, z nichž jedna ukazuje na první a druhá na poslední prvek seznamu.

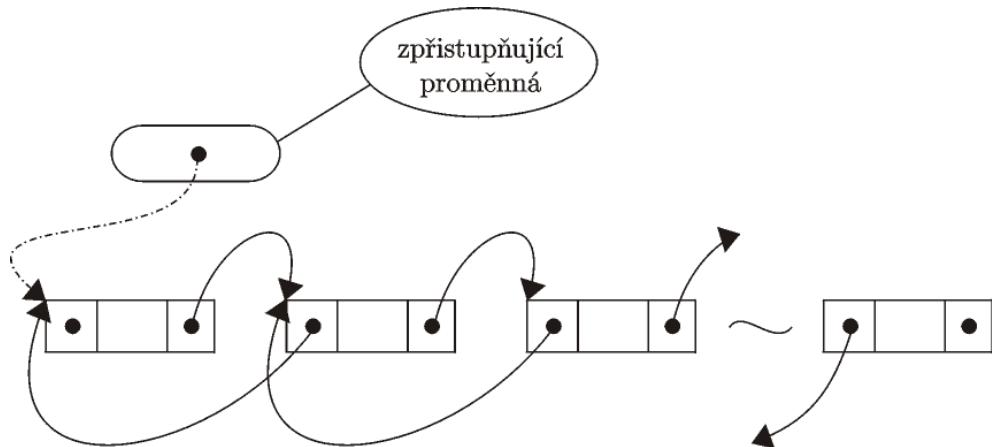


Obrázek 20 - Jednosměrný spojový seznam

Jestliže však potřebujeme zařazovat prvky doprostřed seznamu (nejen na jeho začátek nebo konec) nebo vyřazovat prvky zprostředka seznamu, jsou jednosměrné spojové seznamy značně neefektivní, protože je třeba seznam pracně prohledávat. V takovém případě se doporučuje pracovat s **dvousměrným spojovým seznamem**. Takové seznamy (Obrázek 21) mohou být zpřístupněny jednou či dvěma referenčními proměnnými. Každý prvek seznamu (s výjimkou prvního a posledního) odkazuje nejen na svého následníka, ale také na svého bezprostředního předchůdce.

Kruhový spojový seznam

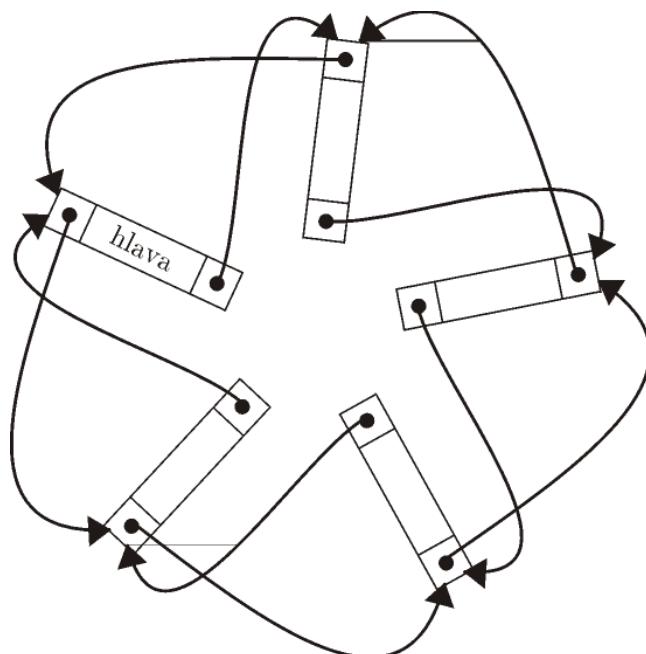
Podstatně efektivnější implementaci kalendáře událostí představuje **kruhový spojový seznam**. Jde v podstatě o speciální případ dvousměrného spojového seznamu, v němž první prvek je definitoricky následníkem posledního a poslední prvek předchůdcem prvního. Každý prvek seznamu bez výjimky má svého předchůdce i následníka a ke zpřístupnění seznamu stačí jediná referenční proměnná.



Obrázek 21 - Dvousměrný spojový seznam

Problémy spojené se zařazováním prvního prvku, resp. s vypouštěním posledního prvku, se elegantně vyřeší tím, že se do seznamu vloží speciální prvek, který se mezi prvky seznamu nepočítá a který se nikdy nevyřazuje. Takový prvek se nazývá **hlava seznamu (head)**. Následníkem tohoto speciálního prvku je první prvek seznamu, jeho předchůdcem prvek poslední (Obrázek 22).

Hlava seznamu



Obrázek 22 - Kruhový spojový seznam (s hlavou seznamu)



Kontrolní otázka:

Zjistěte, jak je realizován kalendář událostí v jazyku SIMULA?

Hierarchický kalendář událostí

7.7 Hierarchické kalendáře událostí

Až dosud jsme předpokládali, že kalendář událostí obsahuje plány všech událostí uspořádané podle jejich aktivačních časů (s přihlédnutím k prioritám). Takové úplné uspořádání není ve skutečnosti nutné, protože vždy hledáme událost, jež má být aktivována jako první, a zpravidla nepotřebujeme znát, která událost je v pořadí druhá, třetí či poslední. Tato skutečnost vedla k myšlence konstruovat **hierarchický kalendář událostí**. Provede se rozklad množiny všech událostí simulačního programu (tzv. prostoru událostí) do několika, přirozeně neprázdných a disjunktních, podmnožin. Události je možno třídit podle prototypu události nebo podle struktury simulačního modelu. Např. pro různé části simulačního modelu vytvoříme samostatné kalendáře. Takové kalendáře mají samozřejmě menší rozsah a práce s nimi je podstatně efektivnější. V každém kroku výpočtu pak můžeme porovnávat jen hodnoty aktivačních časů prvních položek těchto samostatných (lokálních) kalendářů.

Pro hierachický kalendář můžeme také použít reprezentaci lineárního seznamu binárním stromem, což je běžná praxe v pokročilé programovací technice.

7.8 Plánování událostí

Plánování jednotlivých událostí je obecně vázáno na splnění nějaké podmínky, nejčastěji dosažení určité hodnoty simulárního času, ale také dosažení určitého stavu modelovaného systému nebo určité konfigurace časových plánů jednotlivých událostí. V souvislosti s tím se rozlišují:

- **časové plánování** (také imperativní plánování nebo plánování vázaných událostí),
- **podmínkové plánování** (také interrogativní plánování nebo plánování podmíněných událostí).

V prvním případě se testuje pouze dosažení plánovaných hodnot simulárního času, ve druhém pak splnění podmínek týkajících se stavu modelovaného systému či konfigurace časových plánů událostí.

Kalendář událostí, který zajišťuje vedle časového i podmínkové plánování, musí obecně realizovat následující funkce:

1. určit, zda je daná událost naplánována či nikoliv a jakým způsobem je naplánována (časově nebo splněním nějaké jiné podmínky), a v případě, že je skutečně naplánována, specifikovat její aktivační čas, resp. tuto jinou podmínu;
2. vybrat takovou událost z podmínkově plánovaných událostí, jejíž podmínka je splněna (taková událost nemusí existovat);
3. vybrat takovou událost z časově plánovaných událostí, jejíž časový plán aktivace je minimální; existuje-li takových událostí více, vybrat tu s nejvyšší prioritou;

4. naplánovat momentálně nenaplánovanou událost a určit konkrétní čas její realizace, resp. podmínu její realizace;
5. zrušit plán momentálně naplánované události.

Kontrolní úkol:

Vysvětlete význam prostředků standardní třídy SIMULATION jazyka SIMULA (procedury *hold*, *wait* a *passivate*; příkazy **activate** a **reactivate**) při plánování událostí.



Kontrolní otázky:

5. Co je to proces?
6. Jaké jsou vnější stavы procesů?
7. Jak fungují změny stavů procesů?
8. Vysvětlete kalendář událostí. Jak může být implementován?



Korespondenční úkol:

Naprogramujte v jazyku, který znáte (např, Pascal, C, C++) datovou strukturu odpovídající kruhovému dvousměrnému seznamu s hlavou seznamu. Dále naprogramujte procedury (funkce) pro:



- vkládání prvku na konec seznamu,
- vkládání prvku na definované místo v seznamu (za daný prvek a před daný prvek seznamu),
- vyjmání daného prvku ze seznamu,
- vyprázdnění seznamu (kromě hlavy seznamu),
- určení počtu prvků v seznamu,
- určení prvního a posledního prvku seznamu.

Pojmy k zapamatování:



- proces
- vnější stav procesu
 - aktivní
 - ukončený
 - připravený (suspendovaný)
 - pasivní
- vnitřní stav procesu
- kalendář událostí
- spojový seznam
 - jednosměrný
 - dvousměrný
 - kruhový
- hlava seznamu
- plánování událostí
 - časové (imperativní)

- podmínkové (interrogativní)



Shrnutí:

Tato kapitola je věnována výkladu základních principů, které je nutno respektovat při konstrukci simulačního jádra diskrétního modelu. Poznali jste několik nových pojmu jako např. událost, proces s jeho vnějšími a vnitřními stavami, kalendář událostí. S využitím těchto pojmu byste měli pochopit, jak „funguje“ plánování událostí v diskrétním simulačním modelu.

8 Generování pseudonáhodných čísel

V této kapitole se dozvítíte:

- Kritéria náhodnosti
- Algoritmy pro generování pseudonáhodných čísel
- Generování pseudonáhodných čísel z daného rozdělení
- Implementace generátoru
- Testování generátoru

Po jejím prostudování byste měli být schopni:

- vysvětlit význam generátorů pseudonáhodných čísel pro simulaci systémů se zahrnutím náhodných vlivů,
- pochopit princip základních algoritmů pro generování pseudonáhodných čísel,
- otestovat generátor pseudonáhodných čísel implementovaný v programovacích jazycích, které znáte (např. Pascal, C, C++).

Klíčová slova této kapitoly:

Kritérium náhodnosti, pseudonáhodná čísla, generátor.

Doba potřebná ke studiu: 2 hodiny

Průvodce studiem

Principiálně rozlišujeme dva základní způsoby generování náhodných čísel. Prvním z nich je generování pseudonáhodných čísel, druhým generátor pravých náhodných čísel, který vypisuje náhodnost z fyzikálních jevů. V této kapitole se stručně seznámíte se základními algoritmy pro generování pseudonáhodných čísel.



Procesy v reálném světě mají často náhodný charakter. Zjistíme-li při analýze reálného systému náhodný charakter u některého z jeho procesů, musíme tuto skutečnost respektovat i při konstrukci simulačního modelu. Přitom zpravidla vycházíme z experimentálních dat. Na základě těchto dat můžeme

- buď vybrat vhodnou teoretickou distribuční funkci (exaktní vzorec popisující pravděpodobnostní rozdělení dat),
- nebo zkonztruovat empirickou distribuční funkci na základě rozdělení četností uvažovaných dat.

V praxi se hledají algoritmy vytvářející posloupnosti náhodných čísel, která splňují **základní kritéria náhodnosti**. Předpokládejme, že tato čísla jsou ve tvaru pravých desetinných zlomků (jmenovatel je větší než čitatel) s pevným počtem s číslic za desetinnou čárkou, tj. ve tvaru

$$\frac{a_1}{10} + \frac{a_2}{100} + \dots + \frac{a_s}{10^s},$$



kde $a_i \in \{0, 1, \dots, 9\}$, $i = 1, 2, \dots, s$. Číslice a_i pak musí splňovat tyto podmínky:

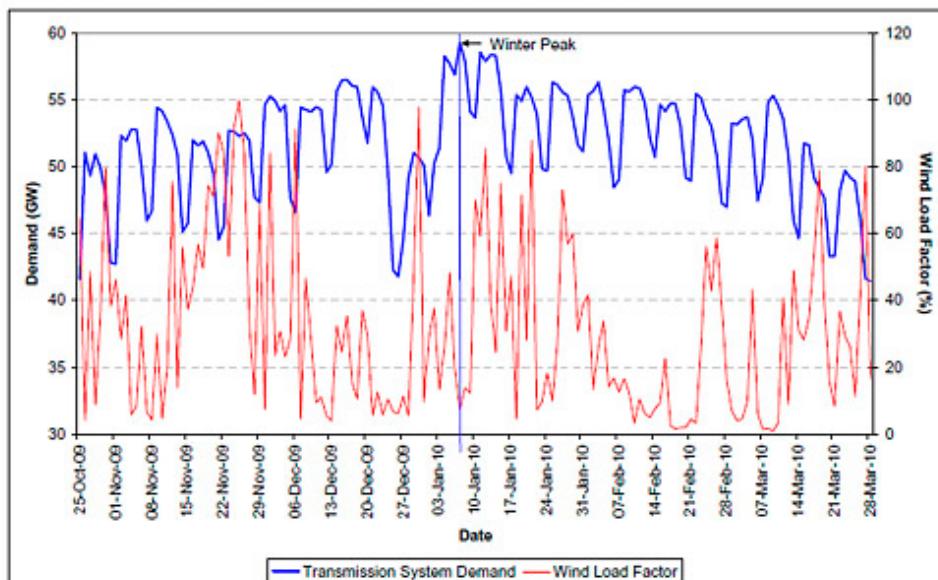
Generátor náhodných čísel

1. každá číslice je vybrána náhodně z množiny $\{0, 1, \dots, 9\}$, přičemž všechny prvky této množiny mají stejnou pravděpodobnost, že budou vybrány;
2. výběr číslice a_i nemá žádný vliv na výběr následující číslice a_{i+1} , $i = 1, 2, \dots, s-1$.

Protože s je konečné, pocházejí takto získaná čísla z rovnoměrného (diskrétního) rozdělení na intervalu $(0,1)$. Je-li však s dostatečně velké, můžeme toto rozdělení považovat za kvazi-spojité.

Náhodná čísla generujeme prostřednictvím tzv. generátoru náhodných čísel. **Generátor náhodných čísel** je takové výpočetní nebo fyzické zařízení, které je určeno ke generování sekvence čísel postrádající jakýkoliv vzor, tj. čísel vypadající jako náhodná. Vstupními parametry generátoru mohou být minimální a maximální generovaná hodnota, případně maximální rozpětí mezi generovanými čísly. Pomocí softwarových metod – speciálních algoritmů jsme schopni generovat tzv. pseudonáhodná čísla. Pomocí hardwarových (fyzikálních) metod jsme schopni generovat pravá náhodná čísla. Generování pomocí fyzikálních metod spočívá v měření určitého fyzikálního děje, např. měření napěťových špiček v síti (Obrázek 23), ale může jí ti o měření vstupů od uživatele (náhodný pohyb myši).

Figure A.30 – 2009/10 Daily Peak and Wind Generation



Obrázek 23 – Měření výkonu větrných elektráren v porovnání s požadavky sítě (http://www.windbyte.co.uk/ims/windpower/ng_winter0910_wind_demand.jpg)



Pseudonáhodné číslo

Posloupnosti náhodných čísel je možno generovat na základě mnohokrát opakovaných, skutečně náhodných experimentů (např. náhodný výběr prvků z množiny $\{0, 1, \dots, 9\}$ s vracením vybraných prvků). V praxi se vytváření posloupnosti náhodných čísel řeší zpravidla softwarově pomocí speciálních algoritmů (viz dále), které v podstatě splňují základní kritéria náhodnosti. Taková čísla se označují jako **pseudonáhodná**, protože posloupnosti generované deterministicky (počítačem na základě daného algoritmu) nemohou být principiálně náhodné.

Doporučujeme Vám, abyste si podrobněji prostudovali standardní prostředky jazyka SIMULA pro tento účel včetně procedury histo pro statistickou analýzu dat.



8.1 Algoritmy pro generování pseudonáhodných čísel

Historicky prvním algoritmem pro vytváření posloupnosti pseudonáhodných čísel je **algoritmus kvadratického středu** (nebo také nazýván jako algoritmus prostředních řádů druhé mocniny), který navrhl John von Neumann. Tento algoritmus je velmi jednoduchý.

von Neumannův algoritmus

0. vyber přirozené číslo o $2k$ číslicích,
1. umocni vybrané číslo na druhou,
2. vyber z mocniny prostředních $2k$ číslic,
3. toto číslo považuj za další prvek posloupnosti,
4. přejdi do bodu 1.

Příklad von Neumannova algoritmu kvadratického středu ($k = 2$)



1. vyberme přirozené číslo o $2k$ číslicích

$$(2013, 4, 11, 0, 5, 133) \rightarrow 2013$$

opakujme body 1. až 4. (pokud nelze vybrat $2k$ číslic ze středu, doplníme před výsledek umocnění vhodný počet nul)

$$\begin{array}{ccc} \dots & \rightarrow & \dots \underline{2013} \dots \\ 2013^2 & \rightarrow & 04\underline{05}2169 \\ 521^2 & \rightarrow & 27\underline{14}41 \\ 7144^2 & \rightarrow & 5103\underline{67}36 \\ 367^2 & \rightarrow & 134\underline{68}9 \\ 3468^2 & \rightarrow & 12027\underline{02}4 \\ 270^2 & \rightarrow & \dots \end{array}$$

Výsledná posloupnost pseudonáhodných čísel – (2013, 521, 7144, 367, 3468, 270, ...).

Nevýhodou popsaného algoritmu je skutečnost, že po dostatečně dlouhé době dochází k vynulování generátoru.

V současnosti jsou zřejmě nejspolehlivější generátory založené na **množstevně kongruentní metodě**. Takové generátory vytvářejí z libovolného základního přirozeného čísla (násady generátoru) $x_0 < m$ posloupnost $\{x_i\}$ přirozených čísel z intervalu $\langle 0, m-1 \rangle$ podle rekurentního vztahu

množstevně kongruentní metoda

$$x_{i+1} = (cx_i + h) \mod m,$$

kde c je násobitel, h přírůstek a m modul, přičemž $m > c$ a $m > h$. Generovaná posloupnost $\{x_i\}$ přirozených čísel je nutně periodická s periodou menší nebo rovnou modulu m . Např. v programovacím jazyku SIMULA se používá rekurentního vztahu

$$U_{i+1} = U_i 5^{2p+1} \mod 2^n,$$

kde p a n jsou vhodně zvolené konstanty. Je-li výchozí U kladné liché číslo, pak jsou kladné a liché také všechny členy generované posloupnosti $\{U_i\}$. Posloupnost je periodická s periodou 2^n a při dostatečně velkém n splňuje velmi dobře základní požadavky kladené na pseudonáhodný výběr.

Kontrolní otázky a úkoly:



1. Vyzkoušejte si alespoň funkci von Neumannova generátoru pro $k = 6$.
2. Nalezněte, jakými prostředky disponuje jazyk SIMULA pro generování pseudonáhodných čísel?
3. Vygenerujte 100 pseudonáhodných čísel z rovnoměrného rozdělení na intervalu $<0, 1>$ pomocí nějakého známého programovacího jazyka a provedete na získaných datech test nezávislosti.

8.2 Generování pseudonáh. čísel z daného rozdělení

Generátory popsaného typu vytvářejí pseudonáhodné posloupnosti $\{x_i\}$, jejichž členy pocházejí z rovnoměrného rozdělení na intervalu $(0,1)$. Pomocí jednoduché lineární transformace

$$y_i = A + (B - A)x_i \quad (i = 1, 2, \dots),$$

se pak generuje pseudonáhodná posloupnost z rovnoměrného rozdělení na intervalu (A, B) .

V praxi často potřebujeme generovat pseudonáhodná čísla z jiného než rovnoměrného rozdělení. Předpokládejme pro jednoduchost, že distribuční funkce požadovaného rozdělení $F(x)$ je spojitá a ryze monotónní, tj. existuje k ní funkce inverzní F^{-1} . V takovém případě platí:

Je-li $\{x_i\}$ posloupnost pseudonáhodných čísel z rovnoměrného rozdělení na intervalu $(0,1)$, pak posloupnost $\{y_i\}$, kde $y_i = F^{-1}(x_i)$, $(i = 1, 2, \dots)$, je tvořena pseudonáhodnými čísly z rozdělení definovaného distribuční funkcí $F(x)$, protože

$$\mathbf{P}(y_i < z) = \mathbf{P}(F^{-1}(x_i) < z) = \mathbf{P}(x_i < F(z)) = F(z).$$

Popsané metody (tzv. **inverzní metody**) nelze ovšem použít v každém případě. Je samozřejmě nepoužitelná tam, kde potřebujeme generovat pseudonáhodná čísla z nějakého rozdělení diskrétního typu (např. binomického nebo Poissonova). Také pro některé spojité distribuční funkce

jsou vyvinuty speciální metody. Tak např. pro generování pseudonáhodných čísel z normálního rozdělení se využívá centrální limitní věty, podle níž má součet dostatečně velkého počtu náhodných veličin s rovnoměrným rozdělením na intervalu $\langle 0,1 \rangle$ přibližně normální rozdělení.

Příklad:

Předpokládejme, že chceme generovat pseudonáhodná čísla y_i normálního rozdělení $N(0, 1)$ pomocí centrální limitní věty. Vyjdeme přitom z $n = 12$ pseudonáhodných čísel x_i s rovnoměrným rozdělením na intervalu $\langle 0,1 \rangle$. Požadovaná čísla budeme generovat pomocí vztahu

$$y_i = \sum_{j=1}^{12} x_j - 6.$$



Spočteme-li střední hodnotu (**E**) a rozptyl (**D**) takto generovaných čísel, dostaneme skutečně

$$\mathbf{E}y_i = 0 \text{ a } \mathbf{D}y_i = 1,$$

jak bylo požadováno.

8.3 Testování generátoru

Před použitím generátoru v simulačních experimentech je nutno ověřit jeho vhodnost, což znamená, že skutečně generuje posloupnost čísel, která splňuje základní podmínky náhodnosti. Před vlastním testováním se generátor upraví tak, aby vytvářel posloupnost pseudonáhodných čísel z rovnoměrného rozdělení na intervalu $\langle 0,1 \rangle$.



Testů navržených pro ověřování kvality generátorů je celá řada. Obecně platí, že je třeba k ověřování generátorů používat více testů, které hodnotí různé vlastnosti generované číselné posloupnosti. Nejčastěji se užívají:

- frekvenční test,
- testy autokorelace,
- sériový test.

Frekvenční test slouží k ověření rovnoměrnosti rozdělení generovaných pseudonáhodných čísel. Základní interval $\langle 0,1 \rangle$ se rozloží na M stejně dlouhých subintervalů $\langle i/M, (i+1)/M \rangle$, $i = 0, 1, \dots, M-1$ a zjišťuje se počet generovaných čísel v jednotlivých subintervalech. Teorie říká, že při dokonale rovnoměrném rozdělení by měly být teoretické četnosti (počty generovaných čísel) ve všech subintervalech stejné. Frekvenčním testem se ověřuje shoda těchto teoretických četností s experimentálními četnostmi získanými při použití generátoru.

Frekvenční test

Testy autokorelace jsou určeny k hodnocení stupně vzájemné korelace (autokorelace) mezi členy posloupnosti generovaných pseudonáhodných čísel. Počítají se autokorelace řádů $1, 2, \dots, m$, přitom se požaduje, aby tyto autokorelace nabývaly co nejmenších hodnot.

Test autokorelace

Sériový test patří k testům kombinovaným, které prověřují současně rovnoměrnost rozdělení i nezávislost. V principu jde o analogii frekvenčního testu. Provádí se frekvenční test ne jednotlivých pseudonáhodných čísel, ale m -tic po sobě jdoucích pseudonáhodných čísel.

Sériový test

8.4 Implementace generátoru pseudonáhodných čísel

V simulačním programu jsou generátory pseudonáhodných čísel realizovány

- jednak procedurou, která provádí výpočet podle zvoleného algoritmu generátoru,
- jednak speciální proměnnou (tzv. **hnízdem generátoru** nebo také **násadou generátoru**), v níž se uchovává rekurentně měněná hodnota pro další volání procedury generátoru.

Jestliže potřebujeme simulovat více navzájem nezávislých náhodných procesů, je vhodné použít více různých hnízd generátoru. Každému procesu se pak přiděluje samostatné lokální hnizdo. Tato lokální hnizda musí být samozřejmě různě inicializována. Např. v jazyku SIMULA se k inicializaci těchto lokálních hnízd používají po sobě jdoucí lichá (kladná) čísla.

Pojmy k zapamatování:

- pseudonáhodné číslo
- von Neumannův algoritmus
- multiplikativně kongruentní metoda
- frekvenční test
- test autokorelace
- sériový test

Kontrolní otázky:

1. Co je to pseudonáhodné číslo?
2. Jaká jsou základní kritéria náhodnosti?
3. Zmiňte alespoň jeden algoritmus pro generování pseudonáhodných čísel.
4. Jak fungují generátory pseudonáhodných čísel?

Shrnutí:

V této kapitol jste se seznámili s problematikou generování pseudonáhodných čísel. Uvědomte si, že bez její znalosti se při algoritmizaci diskrétních modelů s náhodnými efekty rozhodně neobejdete.

9 Základy teorie spojitéhodynamických systémů

V této kapitole se dozvítě:

- Definice spojitého dyn. systému
- Existence a jednoznačnost řešení spojitého dynamického systému
- Stabilita řešení spojitého dynamického systému
- Stabilita řešení lineárních dynamických systémů
- Stabilita řešení nelineárních dynamických systémů

Po jejím prostudování byste měli být schopni:

- získat základní představu o teorii spojitéhodynamických systémů (definice spojitého dynamického systému a jeho řešení, existence a jednoznačnost řešení, stabilita trajektorií podle Ljapunova),
- naučit se pracovat se základními pojmy této teorie (trajektorie systému, fázový portrét, kritické body, periodické trajektorie, systém poruch),
- dokázat posoudit stabilitu lineárních dynamických systémů.

Klíčová slova této kapitoly:

Spojité dynamický systém, stabilita, lineární systém, nelineární systém.

Doba potřebná ke studiu: 8 hodin

Průvodce studiem

Zaměření této kapitoly je výrazně teoretické, proto bude její studium náročnější než studium kapitol ostatních. Základní definice jsou však natolik „průhledné“, že je při troše soustředění pochopíte. Doporučujeme Vám, abyste si osvěžili znalosti o obyčejných diferenciálních rovnicích 1. řádu a jejich soustavách, které jste získali v bakalářském stupni studia. Navíc využijete i poznatků z lineární algebry (maticového počtu) a to při posuzování stability trajektorií lineárních dynamických systémů (vlastní čísla matice, Hurwitzovo kritérium). Některé pasáže této kapitoly (Ljapunovova přímá metoda, stabilita řešení nelineárních dynamických systémů) jsou nepovinné.



9.1 Definice spojitého dynamického systému a jeho řešení

Vyjdeme z rigorózní definice spojitého dynamického systému jakožto modelu nějakého reálného systému se spojitě se měnícími hodnotami atributů.

Spojité dynamický systém je matematická struktura tvořená:

- otevřenou množinou $\Omega \subset \mathbf{R}^n$ (tzv. **stavovým prostorem**),
- množinou $\{f_1, f_2, \dots, f_n\}$ reálných funkcí definovaných na Ω ,
- soustavou obyčejných diferenciálních rovnic 1. řádu



$$\frac{dx_i}{dt} = f_i(x_1, x_2, \dots, x_n), \quad (9.1)$$

kde $i = 1, 2, \dots, n$.

Stavové proměnné

Proměnné x_1, x_2, \dots, x_n se nazývají **stavové proměnné** dynamického systému a rovnice (9.1) **pohybové rovnice** tohoto systému. Přirozené číslo n nazveme **dimenzí** uvažovaného systému.

O stavovém prostoru se předpokládá, že je jistou otevřenou podmnožinou n -rozměrného euklidovského prostoru \mathbf{R}^n . Okamžitý stav dynamického systému je pak reprezentován v tomto stavovém prostoru bodem, jehož kartézské souřadnice odpovídají hodnotám jednotlivých stavových proměnných.

Pro dynamický systém s pohybovými rovnicemi (9.1) platí, že okamžitá rychlosť změny kterékoli stavové proměnné v daném okamžiku nezávisí explicitně na čase, ale pouze na okamžitých hodnotách stavových proměnných. Takovým dynamickým systémům říkáme **autonomní**.

Řešením dynamického systému je množina reálných funkcí $\{x_1(t), x_2(t), \dots, x_n(t)\}$, které mají následující vlastnosti:

- funkce $x_i(t)$, $i = 1, 2, \dots, n$, jsou definovány na nějakém ne degenerovaném otevřeném intervalu $T \subseteq \mathbf{R}$;
- derivace $\frac{dx_i(t)}{dt}$, $i = 1, 2, \dots, n$, jsou také definovány na intervalu T ;
- množina funkcí $\{x_1(t), x_2(t), \dots, x_n(t)\}$ splňuje soustavu diferenciálních rovnic (9.1) v každém bodě $t \in T$.

Trajektorie (orbita)

Fázový portrét

Kritické body

Periodické trajektorie

Křivky $C(t) = \{x_1(t), x_2(t), \dots, x_n(t)\}; t \in T\}$, které jsou určeny partikulárními řešeními daného dynamického systému, se nazývají **trajektorie (orbita)** tohoto systému. Množina všech trajektorií dynamického systému se nazývá jeho **fázovým portrétem**. Mezi trajektoriemi daného dynamického systému zaujímají zvláštní postavení:

1. **kritické body**, pro něž platí $x_i(t) = c_i$, přičemž konstanty c_i jsou řešeními soustavy rovnic

$$f_i(x_1, x_2, \dots, x_n) = 0 \text{ pro všechna } i, i = 1, 2, \dots, n; \quad (9.2)$$

2. **periodické trajektorie s periodou $\tau > 0$** , které splňují pro všechna i , $i = 1, 2, \dots, n$, podmínky

$$x_i(t) \neq x_i(0) \text{ pro } 0 < t < \tau$$

$$x_i(\tau) = x_i(0).$$

9.2 Existence a jednoznačnost řešení spojitého dynamického systému

V teorii spojitých dynamických systémů zaujímají význačné místo takové systémy, u nichž je zaručena existence právě jediného řešení. V následující větě jsou formulovány postačující podmínky pro existenci a jednoznačnost řešení dynamického systému.

Věta 2. Nechť je dán spojitý dynamický systém s pohybovými rovnicemi (9.1). Nechť funkce $f_i(x_1, x_2, \dots, x_n)$, $i = 1, 2, \dots, n$, splňují tyto podmínky:

- jsou spojité a omezené na otevřené množině $\Omega \subset \mathbf{R}^n$, přičemž $|f_i(x_1, x_2, \dots, x_n)| \leq M$;
- vyhovují Lipschitzově podmínce na množině Ω , tj. existují taková reálná čísla $L_1^i, L_2^i, \dots, L_n^i$, že platí

$$|f_i(x_1, x_2, \dots, x_n) - f_i(x'_1, x'_2, \dots, x'_n)| \leq \sum_{j=1}^n L_j^i |x_j - x'_j|.$$

Dále nechť je dán libovolný bod $[x_{10}, x_{20}, \dots, x_{n0}] \in \Omega$ a číslo $t_0 \in \mathbf{R}$. Označme symbolem D vzdálenost tohoto bodu od hranice množiny Ω . Pak existuje právě jediné řešení $\{x_1(t), x_2(t), \dots, x_n(t)\}$ uvažovaného dynamického systému, které je definované a spojité na intervalu

$$(t_0 - D/(M\sqrt{2}), t_0 + D/(M\sqrt{2})), \text{ přičemž } x_i(t_0) = x_{i0} \quad (i = 1, 2, \dots, n).$$

Důkaz této věty nalezne čtenář např. v učebnici (Rosen 1970).

Dynamické systémy, jež vyhovují podmínkám uvedené věty, jsou striktně kauzální (příčinné). To znamená, že každý bod stavového prostoru Ω jednoznačně určuje celou trajektorii systému, právě tu, která jím prochází. Je-li dán počáteční stav takového systému, pak existuje právě jedna možná trajektorie, po níž se systém do tohoto stavu dostal, a také jediný možný způsob dalšího vývoje systému. Dále se budeme výhradně zabývat právě takovými systémy.

Veškeré dosavadní úvahy se týkaly autonomních dynamických systémů. Na první pohled se zdá, že předpoklad o časové nezávislosti chování dynamického systému je příliš restriktivní a nedovoluje nám řešit prakticky významné problémy. V modelech většiny reálných systémů vystupuje explicitně čas a navíc celá řada parametrů (např. rychlostní konstanty nebo vazebné koeficienty), které charakterizují daný systém a jeho dynamické vlastnosti. Zmíněné omezení je naštěstí jen zdánlivé a dynamické systémy s pohybovými rovnicemi (9.1) jsou natolik obecné, že zahrnují jak neautonomní systémy, tak i systémy s jedním nebo více parametry.

9.3 Stabilita řešení spojitého dynamického systému

Stabilita řešení dynamického systému souvisí s odezvou systému na poruchu (změnu počátečního stavu nebo hodnoty nějakého parametru),



speciálně s limitním chováním trajektorií systému pro $t \rightarrow \infty$. Stabilita tedy charakterizuje globální vlastnosti trajektorie systému v přítomnosti poruchy ve vztahu ke globálním vlastnostem odpovídající trajektorie v nepřítomnosti poruchy.

Lokální informaci o odezvě dynamického systému na poruchu nám poskytuje tato věta (Rosen 1970).

Věta 3. Nechť

$$\frac{dx_1}{dt} = f_1(x_1, x_2), \frac{dx_2}{dt} = f_2(x_1, x_2)$$

jsou pohybové rovnice nějakého dynamického systému, který splňuje podmínky věty pro existenci a jednoznačnost řešení. Nechť $\{x_1(t), x_2(t)\}$ je řešení systému takové, že $x_1(t_0) = x_{10}$ a $x_2(t_0) = x_{20}$. Pak existuje takové okolí O bodu $[x_{10}, x_{20}]$, že k danému $\epsilon > 0$ můžeme nalézt $\delta > 0$ tak, že pro body $[\hat{x}_{10}, \hat{x}_{20}]$ splňující nerovnosti $|\hat{x}_{10} - x_{10}| < \delta$ a $|\hat{x}_{20} - x_{20}| < \delta$ platí

$$|\hat{x}_1 - x_1| < \epsilon \text{ a } |\hat{x}_2 - x_2| < \epsilon$$

(Přitom $\{\hat{x}_1(t), \hat{x}_2(t)\}$ značí řešení daného dynamického systému, které prochází bodem $[\hat{x}_{10}, \hat{x}_{20}]$.)

Podle věty 3 je řešení dynamického systému, jež vyhovuje předpokladům věty o existenci a jednoznačnosti řešení, spojitou funkcí počátečních podmínek. Analogicky lze dokázat, že taková řešení jsou spojitou funkcí jednoho nebo více parametrů. Tato tvrzení mají jen lokální platnost, tj. platí za předpokladu, že se omezíme na dostatečně krátký časový interval.

Analýza stability prakticky významných dynamických systémů ukázala, že existují tři základní typy chování systémů, pokud jde o vzájemný vztah mezi porušenými a neporušenými trajektoriemi. Uvažujme trajektorii $C(t) = \{x_1(t), x_2(t), \dots, x_n(t)\}$ procházející bodem $[x_1(0), x_2(0), \dots, x_n(0)]$. Předpokládejme dále, že přítomnost poruchy má za následek změnu počátečních podmínek, tj. přechod k porušené trajektorii $\hat{C}(t) = \{\hat{x}_1(t), \hat{x}_2(t), \dots, \hat{x}_n(t)\}$, která prochází bodem $[\hat{x}_1(0), \hat{x}_2(0), \dots, \hat{x}_n(0)]$.

Stabilita podle Ljapunova

Trajektorie $C(t)$ se nazývá

- **stabilní podle Ljapunova**, jestliže ke každému danému $\epsilon > 0$ existuje takové číslo $\delta \equiv \delta(\epsilon) > 0$, že pro $t \geq 0$ a všechna i ($i = 1, 2, \dots, n$) platí

$$|\hat{x}_i(0) - x_i(0)| < \delta \Rightarrow |\hat{x}_i(t) - x_i(t)| < \epsilon;$$

- **asymptoticky stabilní podle Ljapunova**, jestliže je stabilní podle Ljapunova a kromě toho pro všechna $t > 0$ a všechna i ($i = 1, 2, \dots, n$) platí



$$\lim_{t \rightarrow \infty} |\hat{x}_i(t) - x_i(t)| = 0;$$

- **nestabilní podle Ljapunova**, jestliže existuje takové číslo $\varepsilon > 0$, že ke každému $\delta > 0$ můžeme nalézt aspoň jednu trajektorii $\hat{C}(t)$ a číslo $t_1 = t_1(\delta) > 0$ tak, aby platilo

$$|\hat{x}_i(0) - x_i(0)| < \delta \wedge |\hat{x}_i(t) - x_i(t)| \geq \varepsilon$$

pro aspoň jedno i ($i = 1, 2, \dots, n$).

Je zřejmé, že nějaká trajektorie $C(t)$ může být stabilní, resp. asymptoticky stabilní, jen když určitým způsobem omezíme třídu jí blízkých porušených trajektorií $\hat{C}(t)$. V takovém případě se hovoří o **podmíněné stabilitě**, resp. **podmíněné asymptotické stabilitě**.

Podmíněná stabilita

Nechť $C(t)$ je trajektorie nějakého dynamického systému a S podmnožina jeho stavového prostoru Ω . Trajektorie $C(t)$ je **podmíněně stabilní**, resp. **podmíněně asymptoticky stabilní, vzhledem k množině S** , jestliže vztah porušených trajektorií $\hat{C}(t)$, které procházejí body množiny S , k uvažované trajektorii $C(t)$ odpovídá prvnímu, resp. druhému, případu v předcházející definici stability podle Ljapunova.

Poznámka. Zavedené pojmy stability se týkají odezvy dynamického systému na poruchu způsobenou změnou počátečních podmínek. V odborné literatuře se setkáváme též s pojmem strukturní stability. Tento pojem souvisí s invariancí topologických vlastností trajektorií (topologií fázového portrétu) dynamického systému vůči malým změnám jeho pohybových rovnic.

Zvláštní postavení mezi trajektoriemi dynamického systému zaujmají **kritické (stacionární) body**. Jejich význam při studiu stability spočívá v tom, že obecný problém posouzení stability libovolné trajektorie systému může být převeden na úlohu posouzení stability kritického bodu nějakého jiného, zpravidla podstatně jednoduššího systému.

Kritické body

Uvažujme tedy dynamický systém s pohybovými rovnicemi (9.1) a předpokládejme, že existují jeho řešení typu $x_i(t) = \text{konst.}$, $i = 1, 2, \dots, n$. Je-li počáteční stav systému určen podmínkami $x_i(0) = c_i$, $i = 1, 2, \dots, n$, pak příslušná trajektorie představuje jediný bod ve stavovém prostoru systému, a to bod $[c_1, c_2, \dots, c_n]$. Kritické body systému je možno považovat za degenerované trajektorie. Pro stabilitu kritických bodů platí v plném rozsahu již uvedená definice, takže můžeme rozlišovat stabilní, asymptoticky stabilní a nestabilní kritické body (ve smyslu Ljapunova).

Následující věta viz. (Rosen 1970) ukazuje, jak transformovat problém určení stability libovolné trajektorie daného dynamického systému na problém určení stability kritického bodu.

Věta 4. Nechť dynamický systém s pohybovými rovnicemi (9.1) splňuje předpoklady věty pro existenci a jednoznačnost řešení. Nechť dále

$C(t) = \{x_1(t), x_2(t), \dots, x_n(t)\}$ je nějaká konkrétní trajektorie tohoto systému. Pak lze přejít k novým stavovým proměnným y_1, y_2, \dots, y_n (provést transformaci stavových proměnných) tak, aby platilo:

- pohybové rovnice uvažovaného systému v nových stavových proměnných mají tvar

$$\frac{dy_i}{dt} = F_i(y_1, y_2, \dots, y_n; t), i = 1, 2, \dots, n; \quad (9.3)$$

- trajektorie $C(t)$ se transformuje do bodu $[0, 0, \dots, 0]$ v soustavě nových stavových proměnných;
- počátek je kritickým bodem systému v nových stavových proměnných;
- stabilita počátku v nových stavových proměnných je táz jako stabilita trajektorie $C(t)$.

Důkaz uvedené věty je triviální (Mladov 1966).

Systém poruch

Výsledný dynamický systém s pohybovými rovnicemi (9.3) nemusí být autonomní, což znamená, že okamžitá rychlosť jeho změny může záviset explicitně na čase. Tento dynamický systém se nazývá **systém poruch**. Jde v podstatě o původní dynamický systém reprezentovaný novými stavovými proměnnými $y_i, i = 1, 2, \dots, n$. Právě uvedená věta nám umožňuje omezit se na analýzu stability triviálních řešení $y_1(t) = y_2(t) = \dots = y_n(t) = 0$ takových systémů poruch.

9.4 Stabilita řešení lineárních dynamických systémů

Relativně jednoduché je studium stability lineárních dynamických systémů, jejichž pohybové rovnice lze zapsat ve tvaru

$$\frac{dx_i}{dt} = \sum_{j=1}^n a_{ij} x_j, \quad i = 1, 2, \dots, n, \quad (9.4)$$

přičemž všechny koeficienty a_{ij} jsou reálná čísla. Počátek soustavy souřadnic (stavových proměnných) je evidentně kritickým bodem uvažovaného systému. Stabilita tohoto kritického bodu (triviálního řešení soustavy (9.4)) je určena vlastnostmi čtvercové matice $\mathbf{A} = (a_{ij})$, konkrétně jejími vlastními čísly $\lambda_i, i = 1, 2, \dots, n$.

Věta 8. Nechť je dán lineární dynamický systém s pohybovými rovnicemi (9.4). Nechť $\lambda_i, i = 1, 2, \dots, n$, jsou vlastní čísla matice \mathbf{A} . Pak je kritický bod v počátku

- **stabilní podle Ljapunova**, když pro všechna $i, i = 1, 2, \dots, n$, platí $\operatorname{Re} \lambda_i \leq 0$ a zároveň všechna vlastní čísla s nulovou reálnou částí jsou jednoduchými kořeny charakteristické rovnice matice \mathbf{A} ;
- **asymptoticky stabilní podle Ljapunova**, právě když platí $\operatorname{Re} \lambda_i < 0$ pro všechna $i, i = 1, 2, \dots, n$;



- **nestabilní podle Ljapunova**, když existuje aspoň jedno i , $i = 1, 2, \dots, n$, takové, že platí $\operatorname{Re} \lambda_i > 0$.

Důkaz, jenž je poněkud zdlouhavý, nalezne čtenář v monografii (Mladov 1966).

Případ, kdy vlastní čísla s nulovou reálnou částí jsou vícenásobným kořenem charakteristické rovnice matice \mathbf{A} , vyžaduje podrobnějšího rozboru, viz. (Mladov 1966).

Uvedená věta má základní význam pro posouzení stability uvažovaných lineárních systémů. Z této věty a z vlastností řešení soustav typu (7.4) vyplývá další významné tvrzení.

Věta 9. Jakákoli trajektorie lineárního dynamického systému s pohy-bovými rovnicemi (7.4) je stabilní podle Ljapunova, resp. asymptoticky stabilní podle Ljapunova, právě když je jeho kritický bod v počátku ljakunovsky stabilní, resp. ljakunovsky asymptoticky stabilní.

Pro posouzení asymptotické stability trajektorií dynamického systému s pohybovými rovnicemi (7.4) se často užívá **Hurwitzova kritéria**. Nechť polynom n -tého stupně

$$\lambda^n + h_1\lambda^{n-1} + \dots + h_{n-1}\lambda + h_n$$

Hurwitzovo kritérium



s reálnými koeficienty h_i , $i = 1, 2, \dots, n$, je charakteristickým polynomem matice \mathbf{A} koeficientů soustavy (7.4). Pak trajektorie odpovídajícího dynamického systému jsou asymptoticky stabilní právě tehdy, když všechny hlavní diagonální minory tzv. Hurwitzovy matice

$$\mathbf{H} = \begin{pmatrix} h_1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ h_3 & h_2 & h_1 & 1 & 0 & 0 & 0 & 0 & 0 \\ h_5 & h_4 & h_3 & h_2 & h_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & h_n & h_{n-1} & h_{n-2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & h_n \end{pmatrix}$$

příslušné danému polynomu jsou kladné, tj.

$$\Delta_1 = h_1 > 0, \Delta_2 = h_1 h_2 - h_3 > 0, \dots, \Delta_n = \det(\mathbf{H}) > 0.$$

V praktických aplikacích se setkáváme nejčastěji s dvoudimenzionálními dynamickými systémy, proto považujeme za účelné uvést podrobněji klasifikaci kritických bodů pro takové systémy. Klasifikace kritických bodů těchto systémů je pro nenulová vlastní čísla λ_i , $i = 1, 2$, matice \mathbf{A} uvedena v tabulce 7.1.

Tabulka 7.1: Klasifikace kritických bodů 2-dimenzionálního lin. dynamického systému

Vlastní čísla matice \mathbf{A}		Typ krit. bodu v počátku	Stabilita krit. bodu
Typ	Vztahy		
Reálné	$\lambda_1 = \lambda_2 < 0$	Uzel (propad)	Asympt. stabilní
	$\lambda_1 = \lambda_2 > 0$	Uzel (zdroj)	Nestabilní
	$\lambda_1, \lambda_2 < 0$	Uzel (propad)	Asympt. stabilní
	$\lambda_1 \neq \lambda_2$		
	$\lambda_1, \lambda_2 > 0$	Uzel (zdroj)	Nestabilní
	$f^j(\alpha_0) \neq \alpha_0$		
	$\lambda_1 < 0 < \lambda_2$	Sedlo	Nestabilní
Komplexně sduřené	$\alpha = 0$	Střed	Stabilní
	$\alpha < 0$	Ohnisko	Asympt. stabilní
	$\alpha > 0$	Ohnisko	Nestabilní
$\lambda_1 = \alpha + i\beta$			
$\lambda_2 = \alpha - i\beta$			

Zvláštní zmínky zasluhuje případ, kdy jedna nebo obě vlastní čísla matice \mathbf{A} jsou nulová. Platí následující tvrzení.

1. Jestliže $\lambda_1 = \lambda_2 = 0$, pak je každá trajektorie systému reprezentována jediným bodem ve stavovém prostoru. Trajektorie (body) takového systému jsou buď stabilní (ovšem neasymptoticky), jsou-li všechny prvky matice \mathbf{A} nulové, nebo nestabilní, je-li aspoň jeden z prvků matice \mathbf{A} různý od nuly.
2. Jestliže je pouze jedno z vlastních čísel λ_1, λ_2 matice \mathbf{A} nulové, pak se dvoudimenzionální dynamický systém redukuje na systém jednodimenzionální. Ve stavovém prostoru takového systému existuje nikoli jeden, ale nekonečně mnoho kritických bodů, které určují přímku v tomto prostoru. Tyto kritické body jsou asymptoticky stabilní, resp. nestabilní, podle toho, zda to vlastní číslo matice \mathbf{A} , jež je nenulové, má hodnotu zápornou, resp. kladnou.

Příklad

Určíme stabilitu trajektorií lineárního dynamického systému s pohybovými rovnicemi



$$\begin{aligned}\frac{dx_1}{dt} &= -3x_1 - x_2, \\ \frac{dx_2}{dt} &= x_1 - x_2\end{aligned}$$

v jistém okolí počátku souřadnic.

Charakteristická rovnice matice \mathbf{A} má tvar

$$\lambda^2 + 4\lambda + 4 = 0,$$

takže uvažované trajektorie jsou asymptoticky stabilní. Ke stejnemu závěru dojdeme i pomocí Hurwitzova kritéria. Pro hlavní diagonální minory Hurwitzovy matice

$$\begin{pmatrix} 4 & 1 \\ 0 & 4 \end{pmatrix}$$

totiž platí: $\Delta_1 = 4$ a $\Delta_2 = 15$.

Kontrolní úkol:

Určete stabilitu trajektorií lineárního dynamického systému s pohybovými rovnicemi

$$\begin{aligned}\frac{dx_1}{dt} &= 2x_1 - x_2, \\ \frac{dx_2}{dt} &= x_1 + 2x_2\end{aligned}$$

v okolí triviálního kritického bodu v počátku souřadnic.

9.5 Stabilita řešení nelineárních dynamických systémů



Uvažujme dynamický systém s pohybovými rovnicemi (9.1), který je definován na nějaké otevřené množině $\Omega \subset \mathbb{R}^n$ obsahující počátek a který splňuje předpoklady věty pro existenci a jednoznačnost řešení. Jestliže funkce f_i , $i = 1, 2, \dots, n$, mají derivace všech řádů v každém bodě množiny Ω , můžeme pravé strany pohybových rovnic rozvinout v Taylorovu řadu se středem v počátku

$$f_i(x_1, x_2, \dots, x_n) = f_i(0, 0, \dots, 0) + \sum_{j=1}^n \frac{\delta f_i(0, 0, \dots, 0)}{\delta x_j} x_j + \text{členy vyšších řádů}$$



V dostatečně malém okolí počátku můžeme zanedbat členy druhého a vyšších řádů, takže pohybové rovnice uvažovaného dynamického systému dostanou tvar

$$\frac{dx_i}{dt} = b_i + \sum_{j=1}^n a_{ij} x_j, \quad i = 1, 2, \dots, n,$$

kde $b_i = f_i(0, 0, \dots, 0)$ a $a_{ij} = \frac{\delta f_i(0, 0, \dots, 0)}{\delta x_j}$. Dynamický systém s těmito pohybovými rovnicemi je lineární, ale jeho pohybové rovnice jsou nehomogenní. Nicméně vhodnou transformací stavových proměnných můžeme tyto pohybové rovnice převést na tvar (9.4).

Studium stability nelineárních dyn. systémů je obtížné právě proto, že neexistuje jednoznačný vztah mezi stabilitou kritického bodu nelineárního systému a stabilitou téhož kritického bodu v odpovídajícím linearizovaném systému. V některých případech je však možno na základě analýzy linearizovaného systému učinit cenné závěry o chování systému nelineárního. Pro dvoudimenzionální dynamické systémy platí tato věta (Rosen 1970).

Věta 10. Nechť je dán dynamický systém s pohybovými rovnicemi

$$\frac{dx_1}{dt} = a_{11}x_1 + a_{12}x_2 + \Phi(x_1, x_2), \quad \frac{dx_2}{dt} = a_{21}x_1 + a_{22}x_2 + \Psi(x_1, x_2), \quad (9.5)$$

Nechť jsou dále splněny podmínky:

1. $\Phi(0, 0) = \Psi(0, 0) = 0$ a okolí počátku souřadnic neobsahuje žádné jiné kritické body;
- 2.

$$\lim_{x_1, x_2 \rightarrow 0} \frac{\Phi(x_1, x_2)}{\sqrt{(x_1^2 + x_2^2)}} = \lim_{x_1, x_2 \rightarrow 0} \frac{\Psi(x_1, x_2)}{\sqrt{(x_1^2 + x_2^2)}} = 0;$$

3. vlastní čísla λ_1, λ_2 matice \mathbf{A} jsou obě nenulová a mají také nenulové reálné části.

Pak je chování nelineárního systému s pohybovými rovnicemi (9.5) v okolí počátku stejné jako chování příslušného linearizovaného systému.

Důkaz tohoto tvrzení je uveden v monografii (Rosen 1970). Je třeba zdůraznit, že linearizace je validní pouze v dostatečně malém okolí počátku souřadnic.

Z uvedených vět vyplývá kritérium pro posouzení stability daného kritického bodu obecného dynamického systému s pohybovými rovnicemi (9.1). O stabilitě kritického bodu $[c_1, c_2, \dots, c_n]$ rozhodují vlastní čísla matice \mathbf{D} ,

$$d_{ij} = \frac{\delta f_i(c_1, c_2, \dots, c_n)}{\delta x_j}, \quad i, j = 1, 2, \dots, n.$$

Jednoznačný závěr lze učinit pouze v případě tzv. hyperbolického kritického bodu, kdy žádné vlastní číslo příslušné matice \mathbf{D} nemá nulovou reálnou část. **Hyperbolický kritický bod** je:

1. asymptoticky stabilním uzlem nebo ohniskem, jestliže všechna vlastní čísla matice \mathbf{D} mají zápornou reálnou část;
2. nestabilním uzlem nebo ohniskem, jestliže všechna vlastní čísla matice \mathbf{D} mají kladnou reálnou část;

- nestabilním sedlem, jestliže některá vlastní čísla matice **D** mají kladnou a jiná zápornou reálnou část.

Závěrem se ještě stručně zmíníme o stabilitě uzavřených (periodických) trajektorií. Uvažujme dynamický systém s pohybovými rovnicemi (9.1) a předpokládejme, že jsou splněny předpoklady pro existenci a jednoznačnost jeho řešení. Dále předpokládejme, že $\gamma(t)$ je nějaká uzavřená trajektorie uvažovaného systému s periodou $\tau > 0$. Pak platí následující tvrzení, viz. (Marek a Schreiber 1984, Rosen 1970):

- Uzavřená trajektorie takového systému musí obsahovat aspoň jeden kritický bod ležící uvnitř této trajektorie. Jestliže obsahuje právě jeden kritický bod, pak je tímto bodem buď uzel nebo ohnisko nebo střed.
- Nutnou podmínkou pro existenci uzavřené trajektorie v oblasti A dvoudimenzionálního systému je, aby funkce

$$J(x_1, x_2) = \frac{\delta f_1}{\delta x_1} + \frac{\delta f_2}{\delta x_2}$$

byla buď identicky nulová v oblasti A nebo měnila znaménko v oblasti A (**Bendixonovo negativní kritérium**).

- Kritériem stability uzavřené trajektorie $\gamma(t)$ jsou vlastní čísla tzv. **matice monodromie Δ** ,

$$\Delta_{ij} = \frac{\delta \phi_i}{\delta x_j}, \quad i, j = 1, 2, \dots, n.$$

- kde $\phi: \mathbf{R} \times \mathbf{R}^n \rightarrow \mathbf{R}^n$ je zobrazení takové, že pro každý bod $\mathbf{x} \in \gamma(t)$ platí $\phi(\tau, \mathbf{x}) = \mathbf{x}$.
- Vlastní čísla matice Δ nezávisejí na volbě bodu $\mathbf{x} \in \gamma(t)$. Tato matice má jeden vlastní vektor tečný k uzavřené trajektorii $\gamma(t)$ a jemu přísluší vlastní číslo 1. Jsou-li ostatní vlastní čísla matice Δ v absolutní hodnotě menší než 1, pak je uzavřená trajektorie $\gamma(t)$ asymptoticky stabilní.

Pojmy k zapamatování:

- spojitý dynamický systém a jeho řešení
- stavové proměnné
- pohybové rovnice
- trajektorie (orbita)
- kritický bod
- periodická trajektorie
- stabilita podle Ljapunova
- systém poruch
- Hurwitzovo kritérium



Kontrolní otázky:

- Jak definujeme spojitý dynamický systém?



2. Co je to stavový prostor a stavové proměnné?
3. Co to jsou trajektorie systému?
4. Co to jsou kritické body?

Shrnutí obsahu kapitoly

Σ

V této kapitole jste se seznámili s matematickým pojetím spojitého dynamického systému, poznali příslušnou odbornou terminologii (např. stavový prostor, stavové proměnné, pohybové rovnice, trajektorie) a naučili se posuzovat stabilitu lineárních dynamických systémů podle Ljapunova. Lineární dynamické systémy mají triviální kritický bod v počátku souřadnic a posuzování stability trajektorií v jistém okolí tohoto kritického bodu se převádí na úlohu nalezení vlastních čísel matice \mathbf{A} takového systému. popř. na využití Hurwitzova kritéria.

10 Algoritmizace spojitéch simulačních modelů

V této kapitole se dozvítíte:

- Numerická řešení
- Metody Rungeho-Kutty
- Víceuzlové metody
- Metody pro řešení tuhých (stiff) soustav
- Posouzení metod numerického řešení soustav obyčejných dif. rovnic

Po jejím prostudování byste měli být schopni:

- získat stručný přehled o problematice řešení obyčejných diferenciálních rovnic 1. rádu a jejich soustav,
- naučit se pracovat s takovými pojmy jako numerické řešení, integrační krok, chyby integrační metody a stabilita metody,
- pochopit princip základních metod numerické integrace,
- poznat základní kritéria používaná pro hodnocení metod numerické integrace.

Klíčová slova této kapitoly:

Algoritmizace, metody Rungeho-Kutty, víceuzlové metody, numerická řešení.

Doba potřebná ke studiu: 4 hodiny

Průvodce studiem

Tato kapitola je věnována spojitém simulačním modelům, které jsou zpravidla reprezentovány soustavou diferenciálních a algebraických rovnic. Z hlediska programátorského je základním problémem algoritmizace numerického řešení soustavy diferenciálních rovnic. V této části se pro jednoduchost zaměřujeme pouze na obyčejné diferenciální rovnice 1. rádu a jejich soustavy. Doporučujeme Vám, abyste si předem zopakovali to, co jste se naučili o řešení takových rovnic v bakalářském stupni studia. Soustředte se na pochopení popsaných algoritmů, abyste je dokázali naprogramovat v jazyku SIMULA nebo v některém jiném programovacím jazyku.



Spojité simulační modely jsou charakterizovány tím, že všechny stavové proměnné nabývají hodnot z nějakého nedegenerovaného intervalu (obsahuje nekonečně mnoho bodů) a v průběhu času se mění pouze spojitě. Chování (evoluce) spojitého modelu se popisuje pomocí soustavy algebraických a diferenciálních rovnic, ať už obyčejných nebo parciálních. Numerické řešení algebraických rovnic nepředstavuje zpravidla vážnější problém, pokud máme k dispozici vhodnou univerzální metodu, např. Newtonovu metodu. Naproti tomu však různorodost modelovaných systémů vyžaduje, aby byly programové prostředky pro spojité simulaci vybaveny řadou univerzálních i speciálních metod pro numerickou integraci.



Simulární čas je diskretizován, a to tak hustě, aby numerické řešení diferenciálních rovnic splňovalo požadavky na přesnost výpočtů. Uchovává

se jen okamžitá hodnota simulárního času, tato hodnota se postupně zvyšuje o délku integračního kroku.



Aplikace spojitéh simulačních modelů přináší následující problémy:

- výběr vhodné metody numerické integrace,
- řízení délky integračního kroku s ohledem na požadovanou přesnost řešení úlohy,
- vysoké nároky na spotřebu strojového času.

Typické úlohy řešené pomocí spojitéh simulačních modelů:

- kmitání mechanických systémů,
- řešení elektrických a elektronických obvodů,
- kompartmentové systémy,
- dynamika populací.

Základní informace o numerické integraci poskytují skripta (Rábová et al. 1992), podrobnější poučení monografie (Nekvinda et al. 1976) a (Ralston 1978).

10.1 Základní pojmy

Uvažujme pro jednoduchost soustavu n obyčejných diferenciálních rovnic 1. rádu

$$\frac{dy_i}{dt} = f_i(t, y_1, y_2, \dots, y_n) \quad (10.1)$$

s počátečními podmínkami $y_i = y_{i0}$, kde $i = 1, 2, \dots, n$.

Tuto počáteční (Cauchyho) úlohu budeme zapisovat v maticovém tvaru

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}) \quad (10.2)$$

s počáteční podmínkou $\mathbf{y}(t_0) = \mathbf{y}_0$. Dále předpokládejme, že jsou splněny podmínky existence a jednoznačnosti řešení v oblasti $\Omega \times J$, kde Ω je stavový prostor uvažovaného systému a J nějaký interval reálných čísel.

Numerické řešení

Numerickým řešením počáteční úlohy se rozumí posloupnost $\{y_i\}$ hodnot

$$y_0 = y(t_0), y_1 = y(t_1), \dots, y_k = y(t_k),$$

které odpovídají hodnotám t_i , $i = 1, 2, \dots, k$, nezávisle proměnné t (zpravidla času) v intervalu $\langle t_0, t_k \rangle$. Hodnota výrazu $h = t_{i+1} - t_i$ přitom udává velikost **integračního kroku**. **Exaktní řešení** uvažované počáteční úlohy budeme značit $Y_i = Y(t_i)$, $i = 1, 2, \dots, k$.

Integrační krok

Cílem numerických metod je nalezení numerického řešení. Přitom se požaduje, aby posloupnost $\{y_i\}$ konvergovala pro $h \rightarrow 0$ k exaktnímu řešení $Y(t_i)$.

Rozlišujeme dva základní typy metod řešení soustavy (10.2):

- Metody, kde se hodnoty funkce $f(t, y)$ počítají jen v bodech $[t_i, y_i]$, přičemž y_i je hodnota numerického řešení v bodě $t = t_i$. Do této skupiny patří především tzv. **víceuzlové metody**.
- Metody, kde se hodnoty funkce $f(t, y)$ počítají navíc v bodech ležících mezi jednotlivými body $[t_i, y_i]$, $i = 1, 2, \dots, k$. Zástupci těchto metod jsou především **metody Rungeho-Kutty**.

Víceuzlové metody

Metody Rungeho-Kutty

V případě jednouzlových metod stačí k výpočtu hodnoty y_{n+1} znát pouze hodnotu y_n bezprostředně předcházejícího uzlu. Víceuzlové metody používají k výpočtu hodnoty y_{n+1} k approximaci bezprostředně předcházejících uzlů.

V obou případech je posloupnost hodnot y_i výsledkem postupné extrapolace, přičemž již výchozí hodnoty v každém kroku jsou zatíženy **lokální chybou**.

Lokální chyba

Tato chyba je součtem dvou příspěvků: **chyby metody (truncation error)**, která je způsobena respektováním jen konečného počtu členů Taylorova rozvoje funkce f , a **zaokrouhlovací chyby (rounding error)**, jež souvisí s omezenou délkou slova v paměti počítače. Chyba jednoho kroku pak přirozeně ovlivňuje výsledky kroků následujících. Celková chyba ϵ po realizaci n kroků, tzv. **akumulovaná chyba po n krocích**, je dána vztahem

$$\epsilon = Y_n - y_n.$$

Akumulovaná chyba

Kvalita použité numerické metody se hodnotí podle těchto základních kritérií:



- přesnost metody (velikost lokální chyby a prostředky pro její odhad),
- stabilita metody,
- časová náročnost výpočtu,
- nároky na operační paměť počítače.

Poznámka. Problematika stability numerického řešení soustavy (10.2) přesahuje rámec těchto skript. Zavedeme proto jen pojem absolutní stability. Metoda je **absolutně stabilní** pro daný krok h a danou soustavu diferenciálních rovnic, jestliže chyba vzniklá při výpočtu hodnoty y_n se nezvětšuje při výpočtu následujících hodnot y_k , $k > n$. K vyšetřování absolutní stability se používá testovací rovnice $y' = \lambda y$, kde λ je konstanta (obecně komplexní číslo). Množina hodnot $h = h\lambda$, pro něž je metoda absolutně stabilní, se nazývá **obor absolutní stability** příslušné soustavy.

Stabilita metody

10.2 Metody Rungeho-Kutty

Těmto metodám se také říká **jednouzlové**, protože k výpočtu hodnoty y_{n+1} stačí znát pouze hodnotu y_n v bezprostředně předcházejícím uzlu.



Vychází se obecně ze vztahu

$$\mathbf{y}_{n+1} - \mathbf{y}_n = \sum_{i=1}^p w_i \mathbf{k}_i, \quad (10.3)$$

kde w_i jsou konstanty a

$$\mathbf{k}_i = h\mathbf{f}(t_n + a_i h, \mathbf{y}_n + \sum_{j=1}^{i-1} b_{ij} \mathbf{k}_j)$$

pro $i = 1, 2, \dots, p$, $h = t_{n+1} - t_n$, a_i a b_{ij} jsou konstanty, přičemž $a_1 = 0$. Metoda popsaná vztahem (10.3) se nazývá **p-hodnotová**, protože používá právě p hodnot funkce $\mathbf{f}(t, \mathbf{x})$. Konstanty w_i , a_i , b_{ij} se spočtou tak, aby získané řešení souhlasilo s Taylorovým rozvojem v bodě $[t_n, \mathbf{y}_n]$ až do P -té mocniny integračního kroku h včetně. Taková metoda se pak nazývá **metoda Rungeho-Kutty řádu P** . Pro $p \leq 4$ zřejmě platí $P = p$.

Příklady metod Rungeho-Kutty:

1. Metoda 1. řádu (Eulerova)

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}(t_n, \mathbf{y}_n)$$

2. Příklad metody 2. řádu

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{h}{2}(\mathbf{f}_n + \mathbf{f}(t_n + h, \mathbf{y}_n + h\mathbf{f}_n))$$

Je-li $\mathbf{f}(t, \mathbf{y})$ pouze funkci času t , jde o aplikaci lichoběžníkového pravidla.

3. Příklad metody 3. řádu

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{1}{6}(\mathbf{k}_1 + \mathbf{k}_2 + \mathbf{k}_3),$$

kde

$$\begin{aligned}\mathbf{k}_1 &= h\mathbf{f}(t_n, \mathbf{y}_n), \\ \mathbf{k}_2 &= h\mathbf{f}(t_n + \frac{h}{2}, \mathbf{y}_n + \frac{\mathbf{k}_1}{2}), \\ \mathbf{k}_3 &= h\mathbf{f}(t_n + h, \mathbf{y}_n - \mathbf{k}_1 + 2\mathbf{k}_2).\end{aligned}$$

Je-li $\mathbf{f}(t, \mathbf{y})$ pouze funkci času t , jde v podstatě o použití Simpsonova pravidla.

4. Metody 4. řádu mají obecný tvar

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \sum_{i=1}^4 w_i \mathbf{k}_i,$$

kde

$$\begin{aligned}
 \mathbf{k}_1 &= h\mathbf{f}(t_n, \mathbf{y}_n), \\
 \mathbf{k}_2 &= h\mathbf{f}(t_n + a_2 h, \mathbf{y}_n + b_{21} \mathbf{k}_1), \\
 \mathbf{k}_3 &= h\mathbf{f}(t_n + a_3 h, \mathbf{y}_n + b_{31} \mathbf{k}_1 + b_{32} \mathbf{k}_2), \\
 \mathbf{k}_4 &= h\mathbf{f}(t_n + a_4 h, \mathbf{y}_n + b_{41} \mathbf{k}_1 + b_{42} \mathbf{k}_2 + b_{43} \mathbf{k}_3).
 \end{aligned}$$

Jednotlivé metody 4. řádu se odlišují volbou konstant w_i , a_i , b_{ij} , $i, j = 1, 2, 3, 4$.

K odhadování přesnosti metod Rungeho-Kutty se používá tzv. **metoda polovičního kroku**, tj. dvojí výpočet: jednak s krokem h , jednak s krokem $2h$. Všechny metody Rungeho-Kutty mají ohraničený obor absolutní stability; jeho rozsah se zvětšuje s rostoucím řádem metody. Jejich implementace na počítači je velmi jednoduchá, integrační krok lze libovolně měnit. Mají přibližně stejnou, často i vyšší přesnost než metody prediktor-korektor stejného řádu (viz odstavec 10.3).

Metoda polovičního kroku



10.3 Víceuzlové metody

Numerická metoda se nazývá **k -uzlová**, jestliže k výpočtu hodnoty \mathbf{y}_{n+1} používá právě k approximací bezprostředně předcházejících. Vychází se z obecného vztahu

$$\mathbf{y}_{n+1} = \sum_{i=0}^r a_i \mathbf{y}_{n-i} + \sum_{i=-1}^r b_i \mathbf{y}'_{n-i}, \quad (10.4)$$

kde a_i , b_i jsou konstanty. Ve vztahu (10.4) platí $k = r + 1$.

Uvedené metody nejsou samostartující. Nejprve je nutno spočítat prvních k hodnot některou z metod typu Rungeho-Kutty nebo jinou jednouzlovou metodou.

Metoda definovaná vztahem (10.4) je řádu právě p , jestliže je přesná pro polynomy stupně p . Přitom se vždy požaduje $p \geq 2$. Některé z koeficientů a_i , b_i mohou být rovny nule.

V podstatě se rozlišují tři základní skupiny víceuzlových metod.

1. Je-li $b_{-1} = 0$, pak hodnota \mathbf{y}_{n+1} je vyjádřena jako lineární kombinace již známých approximací \mathbf{y}_i . Takové metody se nazývají **explicitní** (přímé nebo **prediktorového typu**).
2. Je-li ovšem $b_{-1} \neq 0$, pak (8.4) představuje implicitní rovnici pro \mathbf{y}_{n+1} , protože $\mathbf{y}'_{n+1} = \mathbf{y}'_{n+1} = \mathbf{f}(t_{n+1}, \mathbf{y}_{n+1})$, a takovou rovnici lze řešit jen iteračně. Metody této skupiny se nazývají **implicitní** (nepřímé, iterační nebo **korektorového typu**).
3. Kombinací obou předchozích typů vznikají **metody prediktor - korektor**. Jejich princip je jednoduchý. Pro první výpočet (predikci) \mathbf{y}_{n+1}^P se používá formule explicitní. Následně se určí derivace $\mathbf{y}'_{n+1} = \mathbf{f}(t_{n+1}, \mathbf{y}_{n+1}^P)$ a pak se hledané řešení zpřesňuje (koriguje) pomocí implicitní formule (počítá se \mathbf{y}_{n+1}^C). Dvojice metod prediktor-korektor se vybírá tak, aby obě vybrané metody měly stejný řád.

Přímé metody
prediktory

Nepřímé metody
korektory

Metody
prediktor-korektor

Z velkého počtu víceuzlových metod (viz např. (Nekvinda et al. 1976) nebo (Ralston 1978)) uvedeme jen několik málo příkladů. Přitom zavedeme označení $\mathbf{y}'_i = \mathbf{f}_i$ pro všechna i .

1. Adamsovy (Bashforthovy) prediktory

$$\begin{aligned}\mathbf{y}_{n+1} &= \mathbf{y}_n + \frac{h}{2}(3\mathbf{f}_n - \mathbf{f}_{n-1}), \\ \mathbf{y}_{n+1} &= \mathbf{y}_n + \frac{h}{12}(23\mathbf{f}_n - 16\mathbf{f}_{n-1} + 5\mathbf{f}_{n-2}), \\ \mathbf{y}_{n+1} &= \mathbf{y}_n + \frac{h}{24}(55\mathbf{f}_n - 59\mathbf{f}_{n-1} + 37\mathbf{f}_{n-2} - 9\mathbf{f}_{n-3}).\end{aligned}$$

2. Adamsovy korektory

$$\begin{aligned}\mathbf{y}_{n+1} &= \mathbf{y}_n + \frac{h}{12}(5\mathbf{f}_{n+1} + 8\mathbf{f}_n - \mathbf{f}_{n-1}), \\ \mathbf{y}_{n+1} &= \mathbf{y}_n + \frac{h}{24}(9\mathbf{f}_{n+1} + 19\mathbf{f}_n - 5\mathbf{f}_{n-1} + \mathbf{f}_{n-2}).\end{aligned}$$

Implicitní metody jsou sice pracnější než explicitní, ale jsou obecně přesnější a mají lepší vlastnosti, pokud jde o numerickou stabilitu.

Příklad



Řešme Eulerovou metodou diferenciální rovnici

$$\frac{dx}{dt} = t - x$$

s počáteční podmínkou $x(0) = 1$ na intervalu $\langle 0; 0,6 \rangle$ s konstantním krokem $h = 0,1$. Výsledky (s přesností na 3 desetinná místa) jsou uvedeny v tabulce 10.1.

Tab. 10.1

t_n	$x(t_n)$	y_n	$hf(x_n, t_n)$	ϵ_n
0	1,000	1,000	-0,100	0,000
0,1	0,910	0,900	-0,080	0,010
0,2	0,837	0,820	-0,062	0,017
0,3	0,782	0,758	-0,046	0,024
0,4	0,741	0,712	-0,031	0,029
0,5	0,713	0,681	-0,018	0,032
0,6	0,698	0,663		0,035

Exaktní řešení úlohy má přitom tvar $x(t) = 2e^{-x} + x - 1$. Symboly v záhlaví tabulky mají již dříve zavedený význam. V posledním sloupci jsou uvedeny hodnoty akumulované chyby v jednotlivých krocích.

Kontrolní úkoly:

1. Řešte Eulerovou metodou diferenciální rovnici



$$\frac{dx}{dt} = x^2$$

s počáteční podmínkou $x(0) = -4$ na intervalu $\langle 0; 0,6 \rangle$ s krokem $h = 0,1$ a porovnejte numerické řešení s řešením exaktním.

2. Naprogramujte v jazyku SIMULA (nebo v některém jiném programovacím jazyku) algoritmus metody Rungeho-Kutty 2. rádu.

10.4 Metody pro řešení tuhých (stiff) soustav



Soustava obyčejných diferenciálních rovnic typu (8.2) se nazývá **tuhou** (anglicky **stiff**) **soustavou**, jestliže vlastní čísla λ_i její Jacobiho matice

$$\mathbf{J} = \left(\frac{\delta f_i}{\delta x_j} \right), \quad i, j = 1, 2, \dots, n,$$

jsou značně rozdílná. Je zřejmé, že vlastní čísla matice \mathbf{J} závisejí na čase a v průběhu integrace se tedy mění.

Tuhá soustava obyčejných diferenciálních rovnic má tyto vlastnosti:

- $\operatorname{Re}\lambda_j < 0, j = 1, 2, \dots, n$, pro všechny hodnoty t ,
- poměr

$$R = \frac{\max_j |\lambda_j|}{\min_j |\lambda_j|}$$

- je velké číslo (v typických úlohách z praxe řádu 10^3 až 10^6).

Poměr R je charakteristikou soustavy obyčejných diferenciálních rovnic typu (8.2); nazývá se jejím **tuhostním poměrem**.

Tuhostní poměr

U většiny numerických metod stabilita metody vyžaduje omezení integračního kroku h ve tvaru $|h\lambda| < K$, kde K je konstanta charakteristická pro danou metodu a λ je v absolutní hodnotě největší vlastní číslo Jacobiho matice. Pro velké hodnoty λ je tedy nutno volit malý integrační krok.

K řešení tuhých soustav jsou tedy vhodné metody, jejichž oblast absolutní stability je celá levá polorovina, tj. $\operatorname{Re}\lambda h < 0$. Takové metody se nazývají **A-stabilní**. Pro tyto metody platí:

- Explicitní víceuzlové metody nemohou být A-stabilní.
- A-stabilní implicitní metody mohou být nejvýše 2. rádu.

Příklady A-stabilních metod:

1. implicitní metoda 1. rádu (Eulerova metoda)

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}_{n+1},$$

2. implicitní metoda 2. řádu (lichoběžníková metoda)

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{h}{2}(\mathbf{f}_{n+1} + \mathbf{f}_n).$$

Požadavek A-stability je příliš silný, neumožňuje používat víceuzlových metod řádu vyššího než 2. Proto byly zavedeny tzv. **tuhostně stabilní metody**. Základní poučení o těchto metodách nalezne čtenář např. ve skriptech (Rábová et al. 1992).

10.5 Posouzení metod numerického řešení soustav obyčejných dif. rovnic

Na závěr uvedeme několik poznámek k hodnocení jednotlivých numerických metod podle kritérií zavedených v odstavci 10.1.

Přesnost metody. Metody vyšších řádů jsou obecně přesnější (mají menší lokální chybu) než metody nižších řádů téhož typu. U víceuzlových metod jsou metody implicitní přesnější než explicitní.

Stabilita metody. Pro metody Rungeho-Kutty platí, že s rostoucím řádem metody se zvětšuje oblast její absolutní stability. Naproti tomu u Adamsových explicitních metod a metod prediktor - korektor se oblast absolutní stability zmenšuje s rostoucím řádem metody.

Časová náročnost výpočtu. Rychlosť řešení je pro danou hodnotu integračního kroku h nepřímo úměrná řádu metody. Víceuzlové metody jsou při daném h obecně rychlejší než metody jednouzlové.

Nároky na paměť. Tyto nároky jsou obecně přímo úměrné řádu metody. Pro metody téhož řádu platí, že víceuzlové metody jsou náročnější než metody jednouzlové. U víceuzlových metod mají implicitní metody větší nároky na paměť než metody explicitní.

Pojmy k zapamatování:

- numerické řešení soustavy obyčejných lineárních diferenciálních rovnic
- integrační krok
- víceuzlové metody
- metody Rungeho-Kutty
- lokální chyba
- akumulovaná chyba
- stabilita metody
- metoda polovičního kroku
- přímé metody (prediktory)
- nepřímé metody (korektory)
- metody prediktor – korektor
- tuhé (stiff) soustavy

Kontrolní otázky:



1. Vysvětlete pojmy: numerické řešení, integrační krok, chyba a stabilita integrační metody.
2. Jak postupujeme při algoritmizaci spojitéh simulačních modelů? Jmenujte některé typické řešené úlohy.
3. Jaký je rozdíl mezi numerickým a exaktním řešením?
4. Jaký je rozdíl mezi víceuzlovými a jednouzlovými metodami?
5. Jak posuzujeme metody numerického řešení soustav obyč. diff. rovnic?
6. Popište stručně tzv. Eulerovu metodu.

Shrnutí:

V této kapitole distančního textu jste se ve stručnosti seznámili s problematikou numerického řešení obyčejných diferenciálních rovnic 1. řádu a jejich soustav. Poznali jste, že základním problémem algoritmizace spojitého simulačního modelu je výběr vhodné integrační metody a řízení velikosti integračního kroku. Máte k dispozici řadu jednouzlových i víceuzlových integračních metod a základní kriteria k posouzení jejich vhodnosti pro řešení konkrétního úkolu. Při výběru integrační metody se zpravidla volí kompromis mezi jednoduchostí metody (a tedy rychlostí numerického výpočtu) na jedné straně a její přesností a stabilitou na straně druhé.



11 Modely hromadné obsluhy

V této kapitole se dozvíte:

- Úvodní pojmy (sklady, transakce..)
- Kendallova klasifikace systémů hromadné obsluhy
- Příklad modelu hromadné obsluhy v nástroji Witness 2007
- Příklady modelu systému hromadné obsluhy v jazyku GPSS
- Popis modelu v jazyku SIMULA

Po jejím prostudování byste měli být schopni:

- s některými příklady modelů hromadné obsluhy,
- s přístupy k jejich vytváření (programování),
- s termíny, které je třeba v oblasti simulace systémů hromadné obsluhy znát.

Klíčová slova této kapitoly:

Sklady, transakce, generátory, Kendallova klasifikace, Witness, GPSS.

Doba potřebná ke studiu: 4 hodiny



Průvodce studiem

Systémy hromadné obsluhy, tj. systémy, v nichž jsou fronty, se už od počátků číslicové simulace staly jejím důležitým stimulem. Pokud takové systémy mají více front případně navzájem svázaných nějakými pravidly, je číslicová simulace jedinou exaktní metodou k jejich studiu, výzkumu a případně navrhování a plánování. Proto jím věnujeme tuto kapitolu. Základ teorie hromadné obsluhy, resp. teorie front, položil dánský matematik a inženýr Agner Krarup Erlang, který se zabýval problematikou telefonních sítí ve spojitosti s telefonními ústřednami a obsluhou požadavků zákazníků.

11.1 Úvod

Systémy hromadné obsluhy jsou systémy obsahující **fronty**, **prvky**, **kvůli kterým fronty existují, a prvky, které do těchto front vstupují**, a to s více či méně složitými **pravidly**, jak si fronty vybírají a jak přecházejí z jedné fronty do druhé. Takové děje jsou téměř vždy ovlivněny neznámými faktory, které v modelovaných systémech zobrazujeme jako **náhodné jevy**. Složitá interakce těchto náhodných jevů s řazením ve frontách působí ne-překonatelné problémy při analytickém studiu systémů hromadné obsluhy, a tak představují tyto systémy už od konce 50. let dvacátého století typické objekty vyžadující pro své poznání simulaci. Velká většina takových objektů se chápe jako diskrétní systémy, tj. zanedbává se případné spojité chování prvků, když např. přecházejí mezi frontami a když mění svou pozici v nich, stejně jako se zanedbávají případné další spojité procesy.



Teorie hromadné obsluhy se tedy zabývá systémy, ve kterých dochází k procesům obsluhy mezi zákazníky a obsluhujícími centry. Jako příklad takového systému můžeme uvést např. banku, obchod, výrobní linku, ale třeba i křížovatku řízenou světelnými signály. Zákazníkem pak může být



např. člověk, operace, výrobek; obsluhou může být obchod, stroj, instituce, servis atp.

Systém hromadné
obsluhy



Pro toho, kdo slyší termín **systém hromadné obsluhy** poprvé, musíme poznamenat, že to je odborný výraz pro systémy s **frontami**, tj. s posloupnostmi čekajících prvků (nejde tedy o fronty existující např. ve válkách nebo o něco frontálního, jako jsou v meteorologických zprávách frontální poruchy). Vystihují to některé cizí termíny, např. v angličtině, která takové systémy nazývá **queuing systems**, tedy cosi jako frontující systémy, což můžeme volně přeložit jako systémy, v nichž jsou fronty a v nichž se něco s těmito frontami děje. Fronty nemusí být hned nějaké prostorově uspořádané řady lidí, čekajících např. na prodej vstupenek na vystoupení populárního hudebního souboru; fronty jsou obecně uspořádané seznamy s režimem FIFO (first in, first out), při čemž není nutné, aby prvky těchto seznamů byli lidé, nýbrž to mohou být např. kusy materiálu čekající na postupné zpracování ve výrobním procesu, nebo vlaky čekající na povolení vjezdu do dané stanice. Fronty jsou nejdůležitějšími prvky systémů hromadné obsluhy, ale nejsou samy. Tvoří většinou pevné struktury (tak zvané **báze**) spolu s jinými prvky, v prvé řadě s prvky, kvůli nimž samy fronty vznikají (lidově řečeno, prvky, na které se ve frontě čeká). Tyto prvky se nazývají **facility** (podle anglického termínu **facility**) a **sklady** (podle anglických termínů **storage** nebo **store**). Po této pevné struktuře se pohybují **transakce**. Transakce jsou ovšem i prvky, které nejčastěji do front vstupují. V obecném případě má každá transakce atribut zvaný **priorita** a v závislosti na ní se mohou transakce ve frontách předbíhat.

Transakce

Facilita

Sklad

Kapacita

Facilita je prvek, který je v každém okamžiku schopen interagovat nejvíše s jednou transakcí (v profesi systému hromadné obsluhy se častěji říká *obsloužit nejvýš jednu transakci*); takové obsloužení trvá nějakou dobu a během ní ovšem může požadovat obsloužení jiná transakce. Když k tomu dojde, musí tato transakce čekat ve frontě, až se facilita uvolní a až tato transakce přijde na řadu. **Sklad** je podobný facilitě, avšak je schopen na jednou obsloužit více transakcí, maximálně jistý počet, který se nazývá **kapacitou** skladu. Je sice evidentní, že facilita je sklad s kapacitou rovnou jedné, avšak v praxi se mezi facilitami a skladů rozlišuje: facilita může být buď *volná* nebo *obsazená*, kdežto sklad může být také „částečně obsazený“, může mít *k* volných míst a *m* obsazených míst, kde součet *k* a *m* se ovšem rovná kapacitě.

I když nejjintuitivněji chápeme fronty pracující v režimu **FIFO**, není to jediný způsob, jakým mohou prvky přecházet z front do obsluhy. Obsluha může fungovat i v režime **LIFO**, tzn. jako první je obsloužen prvek, který přijde jako poslední. To může být příklad odběru zboží ze skladu, kdy je nejdříve odebrán prvek, který byl naskladněn jako poslední. Obsluha může fungovat také v režimu priorit, tzv. **PRI**, kdy je vybíráno prvek s nejvyšší prioritou, např. oprava důležitého zařízení. Posledním používaným režimem je tzv. **SIRO** (Selection In Random Order), tedy náhodné pořadí.

Úkoly k zamýšlení:

Přemýšlejte a najděte ve svých vzpomínkách nějaké konkrétní příklady na facility a skladů. A odpovězte si také na otázku, zda sklad jakožto termín systémů hromadné obsluhy může být i něco jiného, než sklad, jak ho chá-



peme obecně v češtině. A musí být sklad takto obecně česky chápán, vždy skladem v pojetí systémů hromadné obsluhy?

Kromě facilit a skladu mohou být v bázích i jiné prvky, např. **brána** (anglicky **gate**). Je to jakési zobecnění výtahu ve veřejné budově obsluhovaného lidským operátorem: má jistou kapacitu a začne obsluhovat všechny transakce, které o obsluhu požádaly, teprve když je počet transakcí roven této kapacitě. Zobecnění spočívá v tom, že „gate“ je obecně opravdu jakási brána (anglicky *gate*), před kterou jakoby čekaly prvky systému, jež na ni narazí při provádění svých životních pravidel, a jsou do ní (a dále za ni) vpuštěny, když je splněna jistá podmínka. Avšak takové prvky se nevyskytují často, a tak se v tomto výkladu omezíme jen na transakce, facility, skladы a fronty.

Úkoly k zamýšlení:

Znáte ze svých vzpomínek nějaké jiné brány v uvedeném významu? A přišli byste na další druhy prvků, které by mohly v systémech hromadné obsluhy být (to jest které mohou nějak manipulovat s transakcemi, ale mají pro to jiná pravidla než facility, skladы a brány)? Zkuste takto přemýšlet a po čase pokračujte ve čtení.



Prvky báze systému jsou spojeny „cestami“ transakcí. V profesi systémů hromadné obsluhy se vztah mezi frontou a tím prvkem, kvůli němuž existuje, vyjadřuje předložkou „před“. Říká-li se, že fronta je před facilitou, resp. skladem, znamená to, že z takové fronty vede k příslušné facilitě či skladu přímá cesta, a když je tento prvek (facilita či sklad) obsazen, čeká transakce, která s ním má interagovat, v té frontě. Mezi prvky každé báze je třeba zahrnout také jeden nebo několik **generátorů** transakcí, které reprezentují vstupy transakcí do systému. Takové generátory nejčastěji produkují transakce nezávisle na stavu systému, avšak v některých případech takto reagují na vstup transakce: produkují jakousi její „dceru“ a mohou tedy reprezentovat místa, kde se transakce dělí. V bázi je také jeden nebo více **spotřebitelů** (anglicky **sink**, tedy přesněji odpad, výtok, díra), což jsou prvky, v nichž transakce mizí (čili opouštějí systém). Občas se v bázi vyskytují i prvky, které transakce slučují, tj. nechají zmizet několik transakcí a místo nich generují transakci jednu, jejíž některé atributy jsou kopíemi transakcí, které do takového prvku vstoupily. Ve výrobním procesu si můžeme takto představit montážní krok, tj. spojení několika součástek na jeden produkt, který dále už vystupuje jako jeden celek.

Generátor transakcí

Úkoly k zamýšlení:

Zkuste si vyjasnit aplikaci pojmu vysvětlených v tomto oddíle na nějakém systému hromadné obsluhy, na který si pamatujiete, s nímž jste se přímo či nepřímo setkali, tj. v němž jste reprezentovali nějaký prvek, nejspíše transakci, nebo o němž jste slyšeli či četli, resp. který jste viděli v televizi či v kině. Nápověda: jistě jste někde čekali ve frontě a v tom případě jde jistě o systém hromadné obsluhy. A ještě otázka: bylo by možné, abyste v nějakém vám známém systému hromadné obsluhy reprezentovali facilitu nebo sklad?



Systémy hromadné obsluhy lze modelovat dvěma základními způsoby, a to pomocí **analytických stochastických modelů**, nebo pomocí **simulačních modelů** (Štěrba). Analytické řešení spočívá ve spočtení či odhadnutí

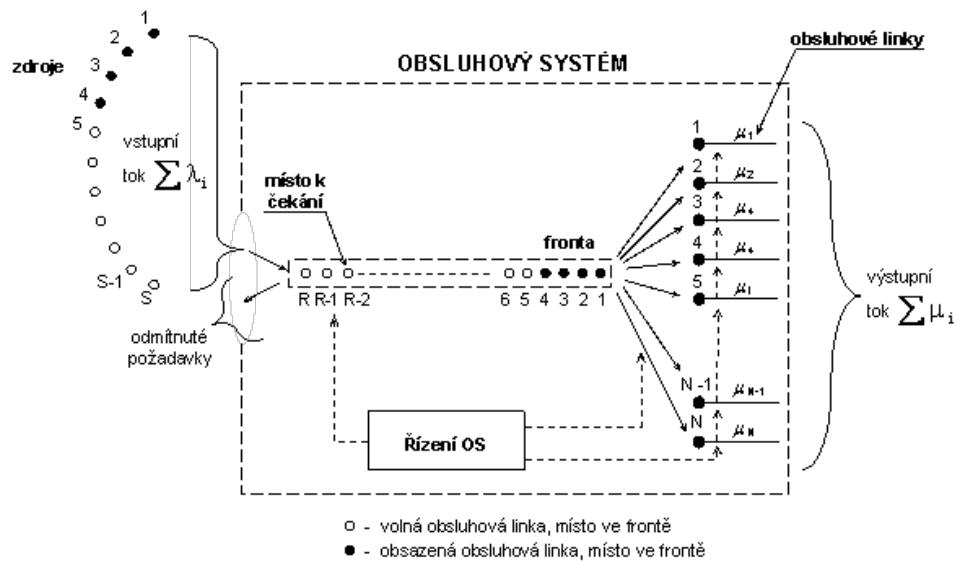


parametrů modelu které nás zajímají, a to na základě známých parametrů modelu. Využíváme při tom nástroje teorie pravděpodobnosti či jiných matematických metod. Obdobně při použití simulačních modelů vyjdeme ze známých parametrů modelu a rovněž odhadneme parametry, které nás zajímají s využitím simulace.

11.2 Klasifikace systémů hromadné obsluhy

K vytvoření modelu hromadné obsluhy je nutné specifikovat (Štěrba):

- vstup jednotek (příchod zákazníků),
- dobu obsluhy na jednotlivých linkách,
- síť obslužných linek (počet, dostupnost...),
- pravidla pro odchod z front do systému obsluhy (FIFO, LIFO...),
- specifické rysy systému (omezení míst ve frontě, netrpělivost zákazníků, atd.).



Obrázek 24 – Schéma jednoduchého obsluhového systému
(http://access.feld.cvut.cz/image/200511161652_image001.gif)

Na (Obrázek 24) můžeme vidět schéma jednoduchého obsluhového systému. Tento model se skládá z N **obsluhových linek**, R **míst určených k čekání požadavků** a z **logiky řízení** obsluhového systému. Požadavky na obsluhu vznikají ve **zdrojích** a tvoří tzv. **vstupní tok** do obsluhového systému. Při vstupu požadavku do systému je řídící logikou rozhodnuto, zda bude požadavek obslužen, či nikoliv. Přijetím daného požadavku je možné započít s jeho **obsluhou**. V případě, že je volná obsluhová linka, je požadavek obslužen, jinak musí ve **frontě** čekat na obslužení. Po obsloužení tento požadavek opouští obsluhový systém a stává se součástí **výstupního toku** již obslužených požadavků.

Pro klasifikaci systémů hromadné obsluhy je nejpoužívanější tzv. **Kendallová klasifikace** (Kendall 1953).

Kendall klasifikuje systémy podle tří hlavních hledisek:

$$A / B / N \quad (11.1)$$

Význam jednotlivých písmen (10.4):

- A – distribuční funkce intervalů mezi příchody
- B – distribuční funkce doby obsluhy
- N – počet obsluhových linek

Tato původní Kendallova klasifikace neurčuje některá další důležitá hlediska systému mezi která patří maximální délka fronty, způsob výběru požadavků z fronty atd. Proto bylo původní Kendallovo značení rozšířeno (FELD 2013):

$$A / B / N / K / S / X \quad (11.2)$$

Oproti původní klasifikaci přibyly tyto symboly:

- K – maximální počet požadavků v systému ($N+R$), nebo počet míst pro čekající (R)
- S – počet zdrojů požadavků
- X – režim fronty

Symbol K není zaveden jednoznačně. Někdy se používá pro označení veškerého počtu požadavků nacházejících se v systému, anebo také jen pro počet volných míst pro čekající požadavky.

Místo jednotlivých symbolů v (11.2) se používají různá značení představující konkrétní hodnoty těchto symbolů. Jde zejména o zavedená označení distribuční funkce mezi jednotlivými příchody, či doby obsluhy, počet linek, počet požadavků, nebo režim fronty. V případě prioritních toků nebo priorit se můžeme setkat s označením R nad daným písmenem (FELD 2013). Můžeme se setkat s množstvím příkladů těchto systémů (Dorda a Teichmann 2012).

Příkladem může být původní Erlangův model (Markovovský model) s upřesněným počtem zdrojů požadavků a režimem fronty:

$$M / M / N / 0 / 4 / FIFO$$



Značící, že používáme exponenciálního rozložení intervalů mezi příchody i exponenciálně rozdelenou dobu obsluhy (M), je použito N obsluhových linek (N), počet míst pro čekající je nula (0), existují čtyři zdroje požadavků (4) a režim fronty je FIFO ($FIFO$).

Pro označení vybrané distribuční funkce se používají i další zavedené zkratky jako např.: D – deterministické (konstantní interval), Ek – Erlangovo rozložení k-tého rádu, Hk – hyperexponenciální rozložení k-tého rádu, GI – nezávislé příchody, IPP – přerušovaný poissonovský proces, atd.



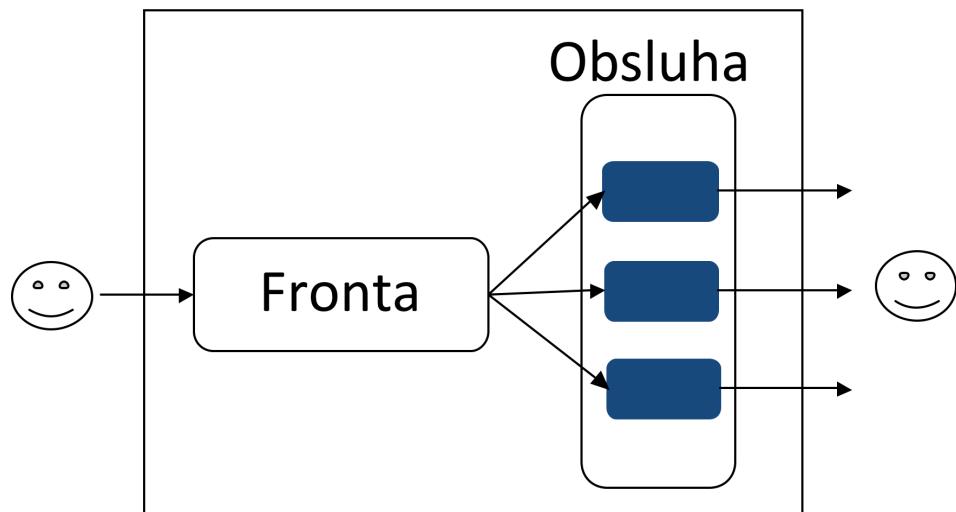
11.3 Simulace systémů hromadné obsluhy v software Witness 2007



Simulační software Witness slouží k simulaci tzv. systémů diskrétních událostí, mezi které také patří systémy hromadné obsluhy. Tento studijní text popisuje tvorbu simulačního modelu jednoduchého systému hromadné obsluhy ve verzi Witness 2007. V novějších verzích Witness se mohou některé zde popisované skutečnosti lišit.

11.3.1 Pojmový model simulovaného systému hromadné obsluhy

Mějme dán systém hromadné obsluhy, který je tvořen 3 paralelně umístěnými obslužnými linkami (Obrázek 25). Doba obsluhy je exponenciální náhodná proměnná s parametrem obsluhy $\mu = 10 \text{ min/zák.}$. Zákazníci přicházejí k systému v Poissonově vstupním toku s intenzitou vstupního toku $0,2 \text{ zák./min.}$. Zákazníci, kteří v okamžiku svého příchodu nenachází žádnou linku volnou, mohou na obsluhu čekat ve frontě, jejíž kapacita je rovna 20. Zákazníci čekající ve frontě jsou obsluhováni v režimu FIFO (řádný frontový režim).



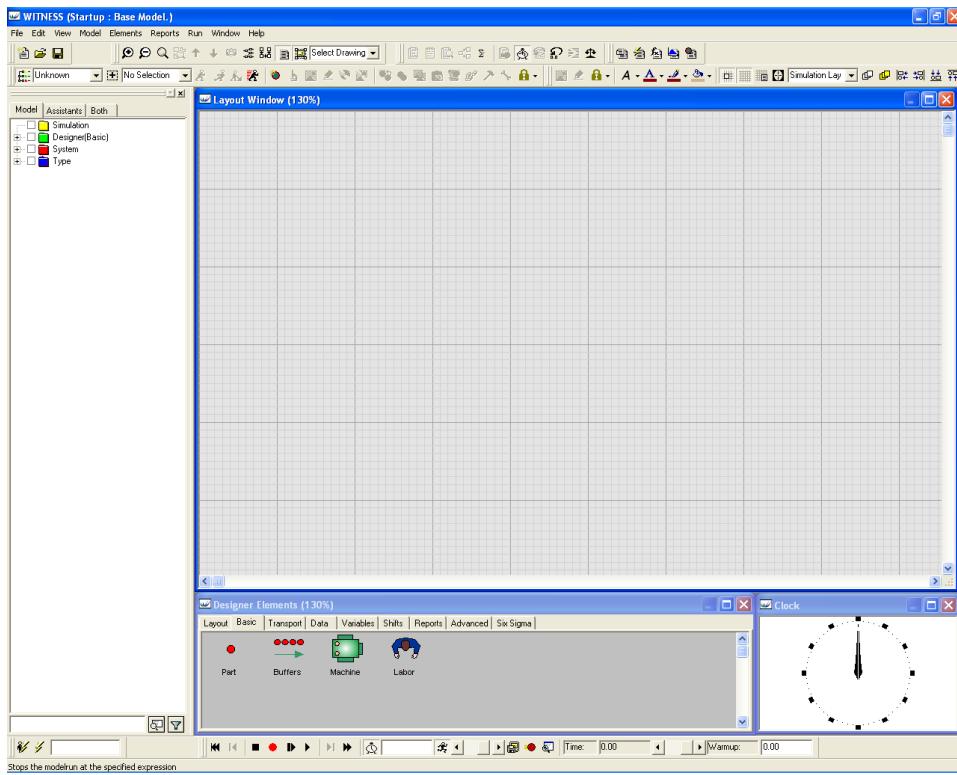
Obrázek 25 - Schéma modelovaného systému hromadné obsluhy

Úkolem je simulací odhadnout základní provozní charakteristiky tohoto systému:

- Střední počet zákazníků v obsluze ES .
- Střední počet zákazníků ve frontě EL .
- Střední počet zákazníků v systému EK .
- Střední doba čekání zákazníka ve frontě EW .
- Střední doba pobytu zákazníka v systému ET .
- Pravděpodobnost odmítnutí zákazníka P_{odm} .
- Využití systému κ .

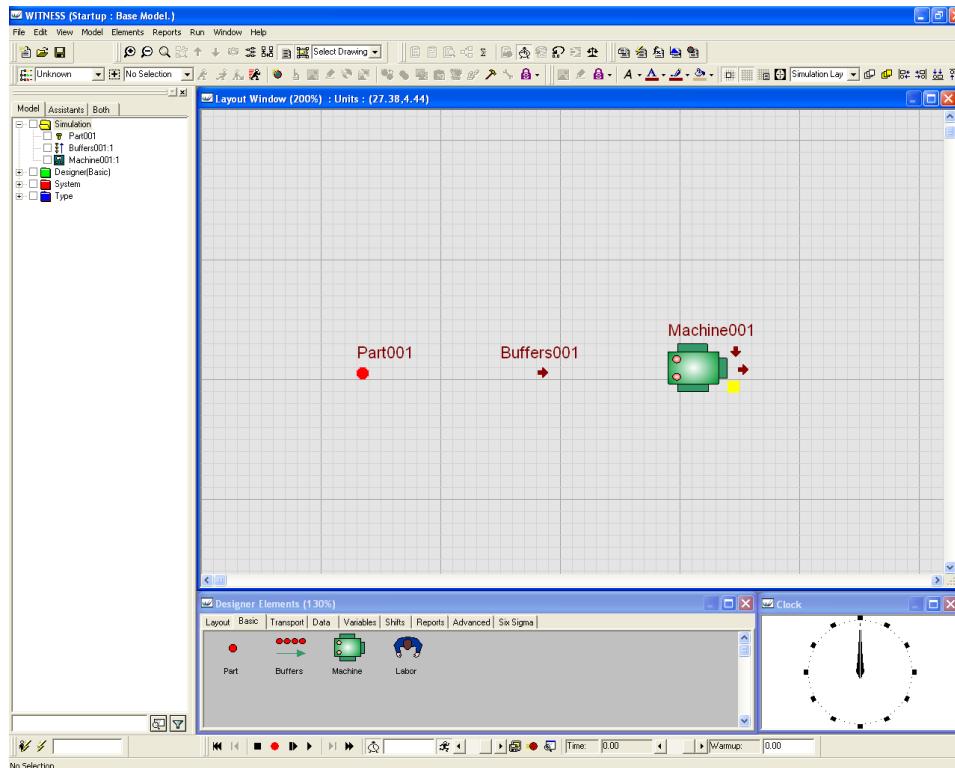
11.3.2 Tvorba simulačního modelu

Nyní musíme pojmový model transformovat do simulačního prostředí Witness. Zákazníky přicházející k systému za účelem obsluhy budeme modelovat pomocí prvku součást („Part“), frontu zákazníků čekajících na obsluhu budeme modelovat pomocí prvku zásobník („Buffers“) a obslužné linky pomocí prvku stroj („Machine“). Všechny tyto prvky nalezneme v záložce „Basic“ okna „Designer Elements“ – (Obrázek 26).



Obrázek 26 – Základní okno Witness 2007

Každý prvek do modelu vložíme tak, že nejdříve klikneme levým tlačítkem myši na vybraný prvek v okně „*Designer Elements*“ a pak dalším kliknutím levého tlačítka myši na vybrané umístění prvku v modelu prvek do simulačního modelu vložíme. Obrázek (Obrázek 27) znázorňuje stav modelu po vložení jednotlivých základních prvků.



Obrázek 27 - Stav modelu po vložení jednotlivých prvků modelu

Pokud chceme přemisťovat prvky po pracovní ploše, můžeme tak činit uchopením prvku stisknutím levého tlačítka myši a držíme-li tlačítko myši a pohybujeme-li kurzorem myši, dochází k přesunu označeného objektu. Při přemisťování prvků je však třeba mít zapnutý režim „*Modul Super Lock*“ zobrazený ikonou  , který nastavíme pomocí ikony visacího zámku („*Toggle Lock Mode*“) v ovládacích panelech. Pokud bychom měli zapnutý režim zámku „*Unlocked*“, nehýbali bychom s celými prvky, ale pouze s označenou částí prvku. Například každá součást je zobrazována dvěma grafickými prvky – jménem a ikonou. Každý stroj je znázorněn celkem 5 grafickými prvky a to:

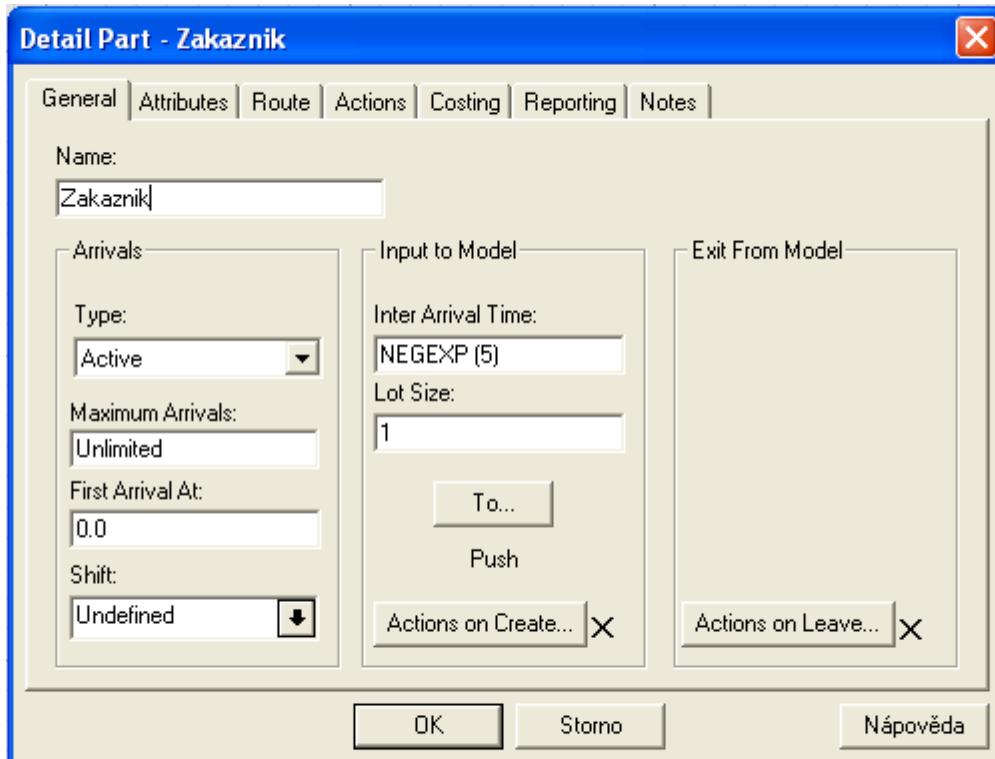
- Jménem stroje.
- Ikonou stroje.
- Prostorem pro zobrazování obsluhy stroje (šipka orientovaná svisle dolů).
- Prostorem pro zobrazení fronty součástí ve stroji (šipka orientovaná vodorovně doprava).
- Zobrazením stavu stroje (vyjádřeno barvou čtverečku, pro nepracující stroj je čtvereček žlutý).

Nyní provedeme postupně základní nastavení jednotlivých prvků. Do nastavení každého prvku se dostaneme dvojklikem levého tlačítka myši na příslušný prvek.

Zmiňme ještě, že časovou jednotku používanou při simulaci si volí uživatel. Zvolme si tedy jako časovou jednotku 1 minutu, je tedy nezbytné všechny časové údaje zadávat v minutách.

11.3.3 Nastavení součásti

Začněme nastavením součásti modelující zákazníky. Výsledné nastavení součásti „*Zakazník*“ je zobrazeno na obrázku (Obrázek 28). V poli označeném „*Name:*“ si součást pojmenujeme, zmiňme, že nelze používat českou diakritiku ani mezery mezi slovy; Witness dále v názvech nerozlišuje velká a malá písmena. Abychom mohli modelovat Poissonovský vstupní tok, je nutné vybrat typ součásti („*Type:*“) jako aktivní („*Active*“). V poli označeném „*Maximum Arrivals*“ můžeme omezit počet součástí daného jména, které mohou do modelu vstoupit. Vidíme, že je přednastaveno „*Unlimited*“, tzn., že proud součástí je teoreticky nekonečný. Omezení počtu vstupujících součástí může být užitečné např. při simulaci uzavřených systémů hromadné obsluhy, pro které je specifické, že počet zákazníků obíhajících v systému je konečný.



Obrázek 28 - Nastavení součásti „Zakazník“

Okno nazvané „First Arrival At:“ slouží k definování hodnoty simulačního času, při jehož dosažení má do modelu vstoupit první součást. Uvažujme, že první zákazník vstoupí do systému v čase 0, proto v tomto poli ponecháme přednastavenou hodnotu.

Důležitým nastavením je nastavení mezer mezi příchody jednotlivých součástí, toto nastavujeme v okně „Inter Arrival Time:“. V této fázi si musíme uvědomit následující fakt. Ze zadání víme, že počet zákazníků, kteří přicházejí k systému za 1 minutu, se řídí Poissonovým rozdělením pravděpodobnosti se střední hodnotou (resp. intenzitou) $\lambda = 0,2$ zák./min. Pro Poissonův vstupní tok platí, že mezery mezi příchody po sobě jdoucích zákazníků se řídí exponenciálním rozdělením se stejným parametrem λ . V okně tedy musíme definovat, že je třeba generovat hodnoty pocházející z exponenciálního rozdělení pravděpodobnosti s příslušným parametrem λ . Hodnoty z exponenciálního rozdělení ve Witness generujeme pomocí funkce:

$$\text{„NEGEXP(EX)“},$$

kde jediným povinným parametrem funkce je střední hodnota EX, která má v našem případě charakter střední mezery mezi příchody po sobě jdoucích zákazníků. Zbývá nám tedy určit tuto střední hodnotu. Jelikož pro střední hodnotu exponenciálního rozdělení platí následující vztah:

$$EX = \frac{1}{\lambda},$$

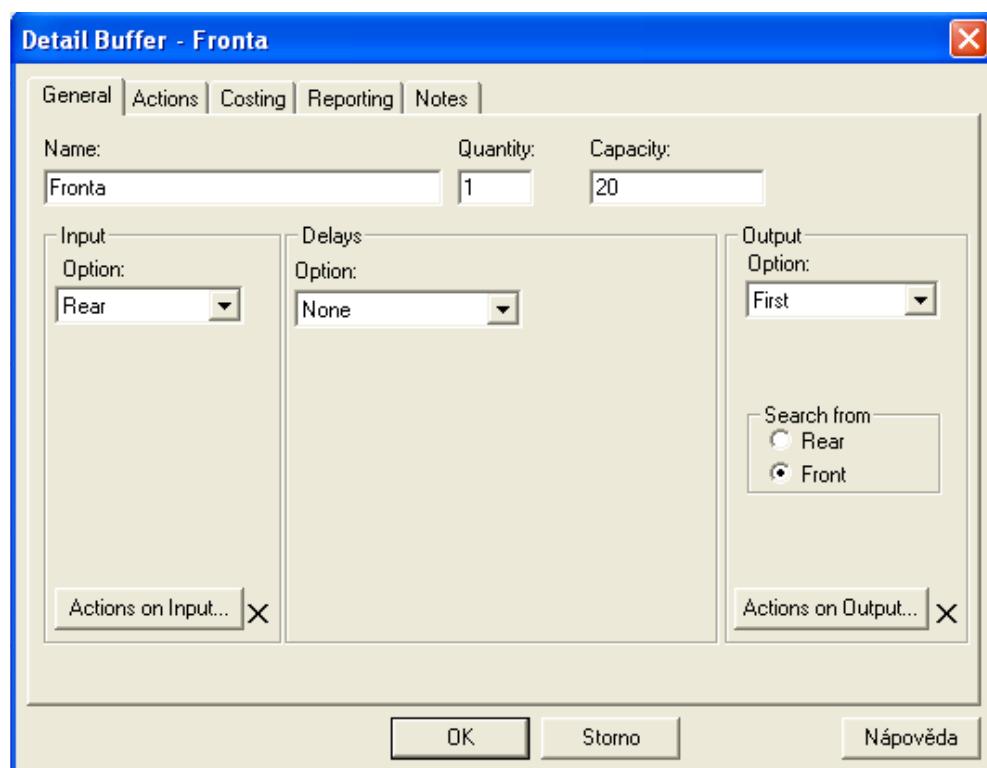
víme, že v našem případě bude střední mezery mezi příchody rovna hodnotě 5 minut. Proto do příslušného okna zapíšeme zápis v podobě:

$$\text{„NEGEXP(5)“}.$$

Poznamenejme, že v případě zadávání desetinného čísla v software Witness **musíme** při jeho zápisu místo desetinné čárky používat desetinnou tečku, čárka se ve Witness používá např. pro oddělování parametrů funkcí.

11.3.4 Nastavení zásobníku

Co se týče nastavení zásobníku (Obrázek 29), který má modelovat frontu zákazníků čekajících na obsluhu, v okně „Name:“ zásobník pojmenujeme „Fronta“. Dalším důležitým nastavením každého zásobníku je jeho kapacita, která se nastavuje v okně „Capacity“. Kapacita zásobníku udává, kolik součástí se může v daném zásobníku současně nacházet. Zde musíme dle zadání nastavit hodnotu 20. Zmiňme ještě, že maximální kapacita, kterou můžeme ve Witness zadat, je shora omezena na hodnotu $2^{16}-1$. Zásobník je v počátečním nastavení nastaven jako FIFO, tudíž není nutno nic dalšího nastavovat. Pokud bychom chtěli specifikovat jiný frontový režim, můžeme tak učinit v nastavení „Input“ a „Output“.



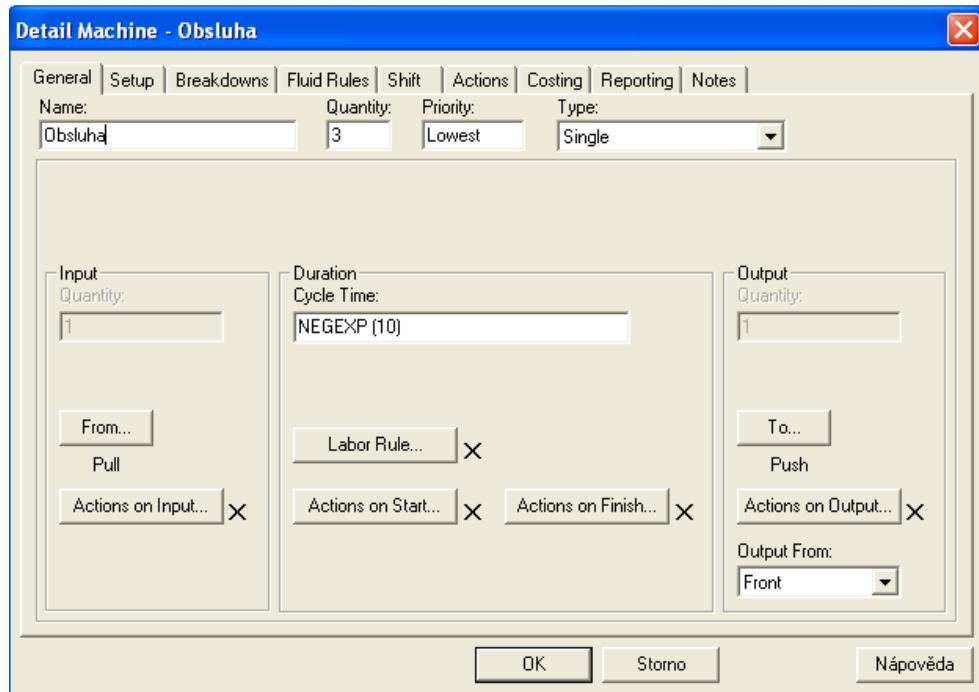
Obrázek 29 - Nastavení zásobníku „Fronta“

11.3.5 Nastavení stroje

Stroj, který jsme si do modelu vložili, nejdříve pojmenujeme jako „Obsluha“ (Obrázek 30). Chceme-li vytvořit více strojů s homogenními parametry a umístěnými paralelně vedle sebe, není nutné vkládat požadovaný počet strojů do modelu. V těchto případech je možno využít nastavení „Quantity“ (nalezneme ho i u zásobníků). Nastavíme-li kvantitu stroje „Obsluha“ na hodnotu 3, Witness automaticky vytvoří 3 paralelně umístěné stroje s homogenními parametry. Dalším důležitým nastavením je nastavení operačního času stroje („Cycle Time:“). Dle zadání víme, že doba obsluhy každého požadavku je exponenciální náhodná proměnná se střední

dobou obsluhy 10 minut. Proto v okně pro nastavení operačního času stroje napíšeme:

,,NEGEXP(10)“.



Obrázek 30 - Nastavení stroje „Obsluha“

Zmiňme ještě, že Witness používá několik typů strojů, které se liší počtem součástí na vstupu a na výstupu ze stroje. Witness 2007 používá např. následující typy strojů:

- Jednoduchý stroj („Single“) – stroj zpracovává pouze 1 součást (na vstupu je tedy 1 součást), po ukončení zpracování součásti ze stroje vystupuje také jenom 1 součást.
- Dávkový stroj („Batch“) – stroj zpracovává součásti po dávkách, na vstupu je n součástí, na výstupu je rovněž n součástí, u dávkového stroje musíme dále definovat minimální a maximální velikost dávky.
- Montážní stroj („Assembly“) – na vstupu do tohoto stroje je n součástí, které stroj montuje do jedné výsledné součásti, na výstupu je tedy pouze 1 součást.
- Produkční stroj („Production“) – jedná se o opak stroje montážního, na vstupu je pouze 1 součást, na výstupu je potom n součástí.
- Obecný stroj („General“) – jedná se o stroj, do kterého vstupuje n součástí a m jich vystupuje.

V našem případě ponecháme jednoduchý stroj, protože na vstupu do stroje je 1 zákazník a na výstupu je rovněž 1 zákazník.

11.3.6 Definování vazeb mezi jednotlivými prvky modelu

V této fázi máme provedeno základní nastavení jednotlivých prvků modelu a nyní je třeba nadefinovat vazby mezi jednotlivými prvky. Tyto vazby definují toky součástí (tedy zákazníků) modelem. Je zřejmé, že nadefinované vazby musí zajistit, že zákazník vstupující do modelu se zařadí do fronty, ze které pak postupuje do obsluhy. Po ukončení obsluhy opouští zákazník model.

K definování těchto vazeb slouží ve Witness tzv. vstupní a výstupní pravidla („*Input Rules*, *Output Rules*“). Při tvorbě jednoduchého modelu systému hromadné obsluhy si vystačíme se 2 základními pravidly:

- Pravidlo „*PULL*“ se používá jako vstupní pravidlo a definuje, že prvek si má vybírat definované součásti z definovaného prvku modelu, např.:
 - „*PULL out of Zasobnik*“ odebere součást z prvku „*Zasobnik*“.
 - Chceme-li specifikovat jméno součásti, která má být odebrána, potom použijeme zápis: „*PULL Soucast out of Zasobnik*“,
- Pravidlo „*PUSH*“ se používá jako výstupní pravidlo a definuje, že prvek má posílat definované součásti do definovaného prvku modelu, logika zápisu je analogická jako u pravidla „*PULL*“.

Začněme nejdříve součástí „*Zakazník*“, která má aktivně vstupovat do modelu a to konkrétně do zásobníku „*Fronta*“. Vstoupíme-li do okna nastavení součásti „*Zakazník*“ a klikneme-li na ikonu určenou pro editaci výstupního pravidla součásti, otevře se nám nové okno, ve kterém požadované pravidlo editujeme. V tomto případě zapíšeme pravidlo ve tvaru:

„*PUSH to Fronta*“.

Poznamenejme, že dosavadní pravidlo „*WAIT*“ je nutné ze zápisu odstranit. Nastavení pravidla lze vidět na obrázku (Obrázek 31).



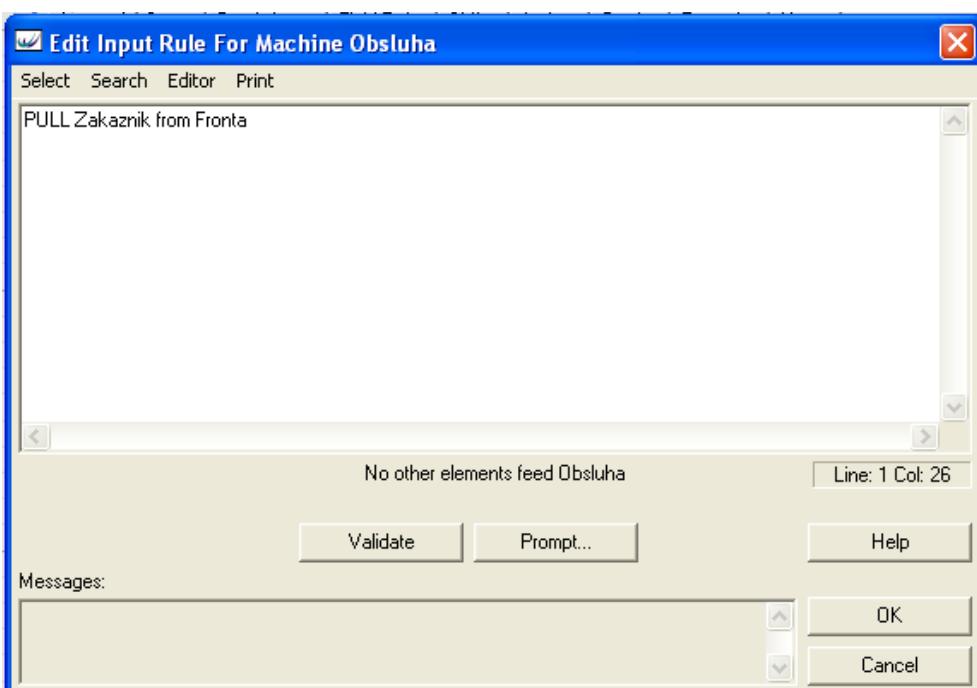
Obrázek 31 - Výstupní pravidlo pro součást „*Zakazník*“

Máme-li toto pravidlo správně definováno, měla by se nám v modelu příslušná vazba zobrazit. Nedošlo-li k jejímu zobrazení, je nutné v nástrojové liště aktivovat ikonu „*Element Flow*“ , čímž dojde k otevření okna, ve kterém je nutno zaškrtnout volbu „*Label Rules*“. Jedná se však pouze o nastavení zobrazování vstupních a výstupních pravidel, funkce pravidla zůstává stejná při nezobrazování i zobrazování pravidla.

Zásobník „*Fronta*“ je při námi definovaném nastavení pasivní, tzn., že u něj nelze editovat ani vstupní, ani výstupní pravidlo. Proto přistoupíme k editaci vstupního a výstupního pravidla pro stroj „*Obsluha*“. Otevřeme si okno s nastavením stroje „*Obsluha*“. Okno pro editaci vstupního pravidla stroje otevřeme kliknutím na ikonu **From...** v okně základního nastavení stroje. Na vstupu do stroje je třeba zajistit, že součást „*Zakazník*“ bude odebírána ze zásobníku „*Fronta*“. Toto zajistíme pravidlem ve tvaru:

„*PULL Zakaznik from Fronta*“.

Detail pravidla definovaného ve Witness můžeme vidět na obrázku (Obrázek 32).

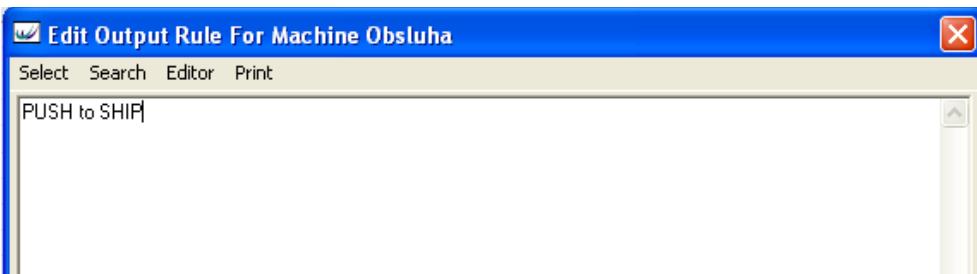


Obrázek 32 - Vstupní pravidlo pro stroj „*Obsluha*“

Po ukončení obsluhy má součást „*Zakazník*“ opustit model, což zajistíme zápisem výstupního pravidla (ikona **To...**) ve tvaru:

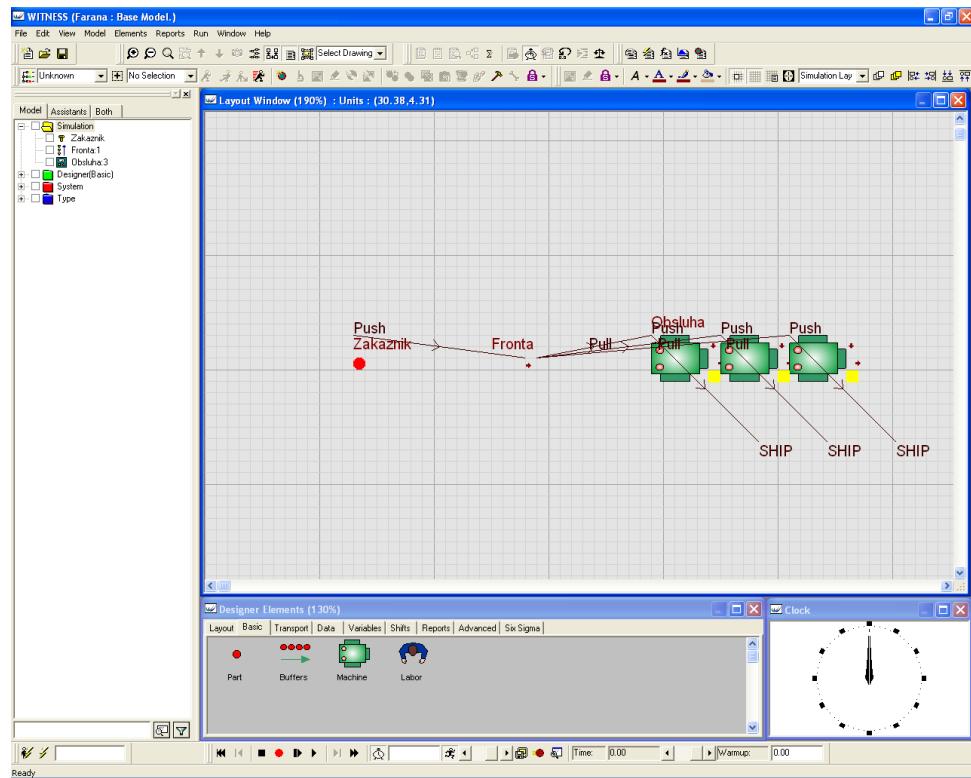
„*PUSH to SHIP*“.

Tento zápis odešle součást „*Zakazník*“ do systémového zásobníku „*SHIP*“ s neomezenou kapacitou, v němž jsou shromažďovány všechny součásti, které byly úspěšně zpracovány a opustily simulační model. Detail nadefinovaného pravidla můžeme vidět na obrázku (Obrázek 33).



Obrázek 33 - Výstupní pravidlo pro stroj „*Obsluha*“

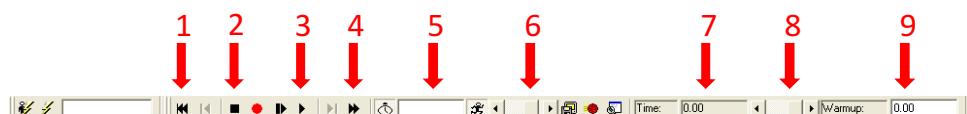
V této fázi je náš model hotov a můžeme přistoupit ke spuštění simulačního pokusu, jehož výstupem budou odhadovány požadované provozní charakteristiky. Hotový simulační model je zobrazen na obrázku (Obrázek 34).



Obrázek 34 - Výsledný simulační model

11.3.7 Spuštění a ovládání simulačního pokusu

Pro spuštění a ovládání simulace slouží spodní panel. Popišme si nyní ovládací prvky, které budeme používat. Tyto prvky můžete vidět na (Obrázek 35).



Obrázek 35 - Panel ovládání simulačního pokusu

Význam ovládacích prvků je následující:

- 1) Pomocí tohoto tlačítka vracíme simulační model do počátečního stavu, tedy na hodnotu simulačního času 0.
- 2) Toto tlačítko slouží pro zastavení simulačního pokusu.
- 3) Toto tlačítko slouží ke spuštění simulace v grafickém režimu.
- 4) Toto tlačítko slouží pro spuštění zrychlené simulace, tedy bez grafického režimu.
- 5) Toto okno slouží pro nastavení simulačního času, po jehož dosažení chceme simulační pokus ukončit. Aby ukončení simulace fungovalo

správně, musí být ikona budíku aktivována a aktuální hodnota simulačního času musí být menší, než činí hodnota, po které chceme simulační pokus ukončit.

- 6) Tento posuvník slouží ke zrychlování a zpomalování pohybu součástí a pracovníků při simulaci v grafickém režimu.
- 7) V tomto okně se zobrazuje aktuální hodnota simulačního času.
- 8) Posuvník slouží ke zrychlování či zpomalování simulačního času při simulaci v grafickém režimu.
- 9) V tomto okně se nastavuje tzv. zahřívací perioda („*Warmup Period*“) na začátku simulačního pokusu. Během této zahřívací periody nedochází k automatickému sběru dat ze simulace, takže zahřívací perioda není zahrnuta do automatického výpočtu simulačních výstupů.

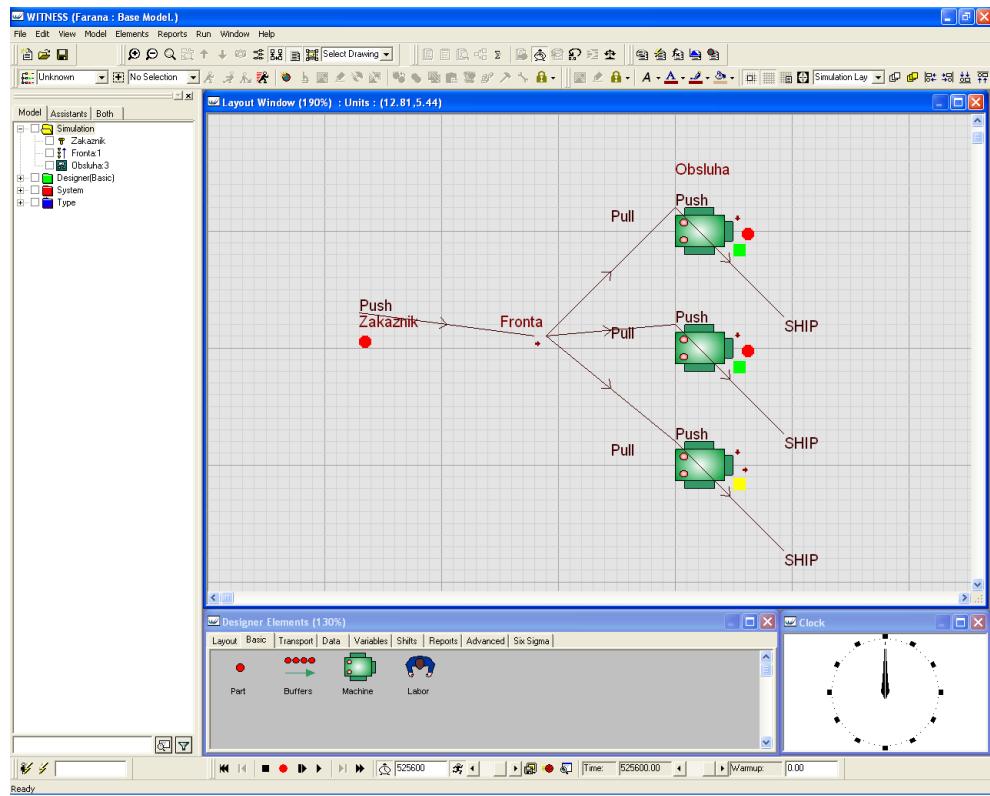
Spustíme 1 simulační pokus s nulovou zahřívací periodou a nastavíme hodnotu simulačního času, při které má být simulační pokus ukončen, na hodnotu 525 600 minut.

11.3.8 Automatické výstupy ze simulačního pokusu

Na (Obrázek 36) vidíme stav simulačního modelu po dosažení hodnoty simulačního času 525 600 minut. Na stejném obrázku jsme grafické zobrazení jednotlivých obslužných linek uspořádali paralelně vedle sebe. Pokud chceme přemisťovat prvky po pracovní ploše, můžeme tak činit uchopením prvku stisknutím levého tlačítka myši a držíme-li tlačítko myši a pohybujeme-li kurzorem myši, dochází k přesunu označeného objektu. Při přemisťování prvků modelu po pracovní ploše lze využívat několik módů.

Máme-li zapnutý režim „*Modul Super Lock*“ zobrazený ikonou  , který nastavíme pomocí ikony visacího zámku („*Toggle Lock Mode*“) v ovládacích panelech, pohybujeme se vsemi částmi vybraného prvku současně. Pokud bychom měli zapnutý režim zámku „*Unlocked*“, nehýbali bychom s celými prvky, ale pouze s označenou částí prvku. Při rozmísťování strojů tak, aby byly zobrazeny paralelně vedle sebe, s úspěchem využijeme režim „*Individual Lock*“.

Při spuštění simulace dochází k automatickému sběru simulačních výsledků pro prvky, u kterých to má smysl – např. součásti, stroje či zásobníky. K této simulačním výstupům se dostaneme, klikneme-li pravým tlačítkem myši na vybraný prvek modelu a v nabídce, která se nám otevře, vybereme volbu „*Statistics*“. Podívejme se nyní na statistiky vybraných prvků.



Obrázek 36 - Stav modelu po ukončení simulačního pokusu

Obrázek (Obrázek 37) zobrazuje automaticky vytvářenou statistiku prvku „Zakazník“. Podívejme se nyní na význam jednotlivých údajů:

- „No. Entered“ – počet součástí „Zakazník“ vstoupivších do modelu.
- „No. Shipped“ – počet součástí „Zakazník“ odeslaných z modelu („SHIP“).
- „No. Scrapped“ – počet součástí „Zakazník“, které byly odeslány do odpadu („SCRAP“).
- „No. Assembled“ – počet součástí „Zakazník“, které byly v modelu smontovány.
- „No. Rejected“ – počet součástí „Zakazník“, které byly odmítnuty na vstupu do modelu.
- „W.I.P.“ – počet součástí „Zakazník“ momentálně se nacházejících v modelu.
- „Avg W.I.P.“ – počet součástí „Zakazník“ průměrně se nacházející v modelu.
- „Avg Time“ – čas, který průměrně stráví součást „Zakazník“ v modelu.

WITNESS

Part Statistics Report by On Shift Time

Name	Zakazník
No. Entered	105598
No. Shipped	105596
No. Scrapped	0
No. Assembled	0
No. Rejected	11
W.I.P.	2
Avg W.I.P.	2.96
Avg Time	14.72
Sigma Rating	6.00

Swap rows and columns Detailed Report

 Group Jobs

Reporting Mode: As Specified Individual Group

Obrázek 37 - Statistiky součásti „Zakazník“

Jako další si vysvětlíme, jaké číselné charakteristiky nám Witness stanovuje pro zásobníky, viz (Obrázek 38).

WITNESS

Buffer Statistics Report by On Shift Time

Name	Fronta
Total In	105598
Total Out	105598
Now In	0
Max	20
Min	0
Avg Size	0.94
Avg Time	4.70
Avg Delay Count	
Avg Delay Time	

Obrázek 38 - Statistiky zásobníku „Fronta“

Witness nám automaticky generuje hodnoty těchto statistik:

- „Total In“ – celkový počet všech částí, které do zásobníku během simulačního pokusu vstoupily.

- „*Total Out*“ – celkový počet součástí, které během simulačního pokusu ze stroje vystoupily.
- „*Now In*“ – počet součástí aktuálně se nacházejících v zásobníku.
- „*Max*“ – maximální počet součástí nacházejících se současně v zásobníku během simulačního pokusu.
- „*Min*“ – minimální počet součástí nacházejících se současně v zásobníku během simulačního pokusu.
- „*Avg Size*“ – průměrný počet součástí nacházejících se v zásobníku.
- „*Avg Time*“ – průměrná doba, kterou stráví každá součást v zásobníku.

Podívejme se nyní na význam vybraných statistik vztahujících se ke strojům v simulačním modelu (Obrázek 39):

- „% *Idle*“ – vyjadřuje podíl času v [%], kdy stroj nepracoval.
- „% *Busy*“ – vyjadřuje podíl času v [%], kdy stroj aktivně pracoval.
- „% *Blocked*“ – vyjadřuje podíl času v [%], kdy byl stroj na výstupu blokován (tzn., že byl ukončen pracovní cyklus stroje, ale nemohlo dojít k odeslání zpracované součásti, resp. součásti pryč ze stroje).
- „% *Cycle Wait Labor*“ – vyjadřuje podíl času v [%], kdy stroj nepracoval z důvodu čekání na jeho obsluhu.
- „% *Setup*“ – vyjadřuje podíl času v [%], kdy byl stroj v aktivní údržbě.
- „% *Setup Wait Labor*“ – vyjadřuje podíl času v [%], kdy stroj čekal na údržbu z důvodu čekání na pracovníka (příp. pracovníky) údržby.
- „% *Broken Down*“ – vyjadřuje podíl času v [%], kdy byl stroj v poruše.
- „% *Repair Wait Labor*“ – vyjadřuje podíl času v [%], kdy stroj čekal na pracovníka (pracovníky) provádějící jeho opravu.
- „*No. Of Operations*“ – číslo vyjadřuje, kolik pracovních cyklů stroj dokončil.

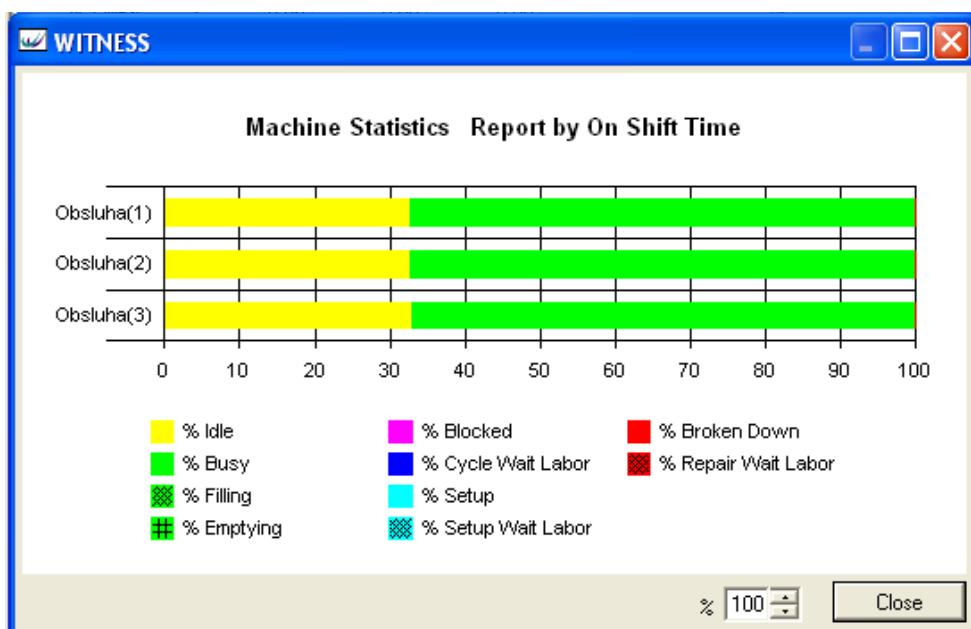
WITNESS

Machine Statistics Report by On Shift Time

Name	Obsluha(1)	Obsluha(2)	Obsluha(3)
% Idle	32.76	32.80	32.97
% Busy	67.24	67.20	67.03
% Filling	0.00	0.00	0.00
% Emptying	0.00	0.00	0.00
% Blocked	0.00	0.00	0.00
% Cycle Wait Labor	0.00	0.00	0.00
% Setup	0.00	0.00	0.00
% Setup Wait Labor	0.00	0.00	0.00
% Broken Down	0.00	0.00	0.00
% Repair Wait Labor	0.00	0.00	0.00
No. Of Operations	35172	35117	35307

Obrázek 39 - Statistiky stroje „Obsluha“

Co se týče výše uvedených procentuálních hodnot, je možno je nechat zobrazit v podobě grafu. Ten získáme kliknutím na tlačítko „Chart States“ v okně zobrazujícím číselné statistiky. Tyto grafické výstupy znázorňuje (Obrázek 40). Jednotlivé stavů jsou rozlišeny barevně, kde žlutá část odpovídá stavu, kdy stroj nepracuje, a zelená část, kdy stroj aktivně pracuje.



Obrázek 40 - Graf stavů stroje „Obsluha“

11.3.9 Odhad požadovaných provozních charakteristik systému

Všechny námi požadované provozní charakteristiky můžeme stanovit na základě automaticky generovaných charakteristik, které jsme prezentovali v předchozí části textu.

- Střední počet zákazníků v obsluze ES – tuto hodnotu můžeme odhadnout, sečteme-li hodnoty „% Busy“ ve statistikách stroje „Obsluha“. Tento součet musíme vydělit hodnotou 100.

- Střední počet zákazníků ve frontě EL – tato hodnota může být odhadnuta na základě hodnoty „Avg Size“, kterou najdeme ve statistikách zásobníku „Fronta“.
- Střední počet zákazníků v systému EK – tuto hodnotu odhadneme na základě hodnoty „Avg W.I.P.“, kterou najdeme ve statistikách součásti „Zakazník“. Další možný způsob výpočtu lze realizovat na základě znalosti předchozích dvou provozních charakteristik a vztahu $ES + EL = EK$.
- Střední doba čekání zákazníka ve frontě EW – tuto hodnotu můžeme odhadnout na základě hodnoty „Avg Time“, kterou najdeme ve statistikách zásobníku „Fronta“.
- Střední doba pobytu zákazníka v systému ET – k odhadu této hodnoty lze použít hodnotu „Avg Time“, kterou najdeme ve statistikách součásti „Zakazník“.
- Pravděpodobnost odmítnutí zákazníka P_{odm} – tuto hodnotu odhadneme na základě znalosti počtu zákazníků, kteří do modelu vstoupili („No. Entered“ ve statistikách součásti „Zakazník“), a počtu zákazníků, kteří byli na vstupu do modelu odmítnuti („No. Rejected“ ve statistikách součásti „Zakazník“). Pravděpodobnost odmítnutí potom odhadneme jako podíl počtu odmítnutých zákazníků a počtu všech zákazníků (součet vstoupivších a odmítnutých zákazníků).
- Využití systému κ – tuto hodnotu odhadneme tak, že spočítáme aritmetický průměr všech hodnot „% Busy“ ve statistikách stroje „Obsluha“.

Odhady všech požadovaných provozních charakteristik jsou uvedeny v (Tabulka 2). Je nutné zdůraznit, že se jedná pouze o bodový odhad provedený na základě jednoho simulačního pokusu. V praxi se však používají intervalové odhady, k jejichž získání bychom museli realizovat více simulačních pokusů a na základě výsledků těchto simulačních pokusů bychom požadované intervalové odhady spočítali.

Tabulka 2 - Přehled dosažených výsledků

Provozní charakteristika	Bodový odhad	Jednotka
ES	$(67,24+67,20+67,03)/100=2,01$	[-]
EL	0,94	[-]
EK	2,96	[-]
EW	4,70	[min]
ET	14,72	[min]
P_{odm}	$11/(105598+11)=0,0001$	[-]
κ	$(67,24+67,20+67,03)/3=67,16$	[%]

11.3.10 Závěr

V tomto krátkém příkladu jsme si ukázali, jak modelovat a simulovat jednoduché systémy hromadné obsluhy v simulačním prostředí WITNESS 2007. Máme-li simulační model vytvořen, můžeme přistoupit ke zkoumání, co by se např. stalo, kdybychom zrušili jednu obslužnou linku. To už ale necháváme na čtenáři.

11.4 Příklad modelu systému hromadné obsluhy v jazyku GPSS



Uvažujme systém hromadné obsluhy znázorněný na obr. 9.1, kde Q_i značí frontu, Z_i facilitu, S_i sklad a G_i vstup do systému. Předpokládejme pro jednoduchost, že generátor G_1 posílá do systému jednu transakci přesně každou desátou minutu a že doby mezi dvěma následujícími vstupy transakcí generátorem G_2 mají normální rozdělení se střední hodnotou pět minut a směrodatnou odchylkou 3 minuty (což budeme dále zkracovat jako $N(5,3)$), avšak že první transakce vstoupí prvkem G_2 až po 30. minutě. Předpokládejme dále, že doby obsluhy mají také normální rozdělení, a to ve facilitách Z_1 , Z_2 a Z_3 po řadě $N(6, 3)$, $N(5, 2)$ a $N(4, 2)$ a ve skladech S_1 a S_2 po řadě $N(40,15)$ a $N(50,20)$. Kapacita skladu S_1 nechť je 10 a kapacita skladu S_2 nechť je 12, po zpracování ve facilitě Z_3 nechť transakce opustí systém a nechť je simulační pokus ukončen, když generátorem G_1 vstoupí do systému tišíc transakcí a je zpracováno a když také generátorem G_2 vstoupí šest set transakcí a je zpracováno.

První implementace amerického jazyka **GPSS** (General Purpose Systems Simulator) (Kindler 1980, Schriber 1991), zaměřeného na simulaci systému hromadné obsluhy, se datuje od roku 1961, takže jde o první jazyk pro diskrétní simulaci na světě. Používá se však dodnes, zejména v USA a v Německu.

Ukážeme, jak se v něm uvedený systém popíše.

```
S1 STORAGE 10
S2 STORAGE 12
GENERATE 10,,0,1000
SEIZE Z1
ADVANCE 6,3
RELEASE Z1
ENTER S1
ADVANCE 40,15
LEAVE S1
SEIZE Z2
ADVANCE 5,2
RELEASE Z2
ENTER S2
ADVANCE 50,20
LEAVE S2
SEIZE Z3
ADVANCE 4,2
RELEASE Z3
TERMINATE
```

```

GENERATE 5,3,30,600
ENTER S2
ADVANCE 50,20
LEAVE S2
SEIZE Z3
ADVANCE 4,2
RELEASE Z3
TERMINATE

```

Vidíme, že jazyk GPSS vede svého uživatele k tomu, že rozpoznává dva různé „druhy životů“ transakce, vlastně dvě různé „třídy“ transakcí, které jsou zajímavé tím, že společně žádají o obsluhu jisté facility a sklady. Facility nemusí uživatel vůbec speciálně zavádět, stačí, když její jméno uvede alespoň jednou v životních pravidlech nějaké třídy transakcí. Sklady explicitně zavést musí, a to kvůli určení jejich kapacity.

Životní pravidlo `SEIZE X` stanoví, že transakce požadá facility X o obsluhu – pokud je tato facilita obsazena, čeká transakce automaticky ve frontě „před“ ní. Životní pravidlo `ADVANCE A,B` stanoví, že transakce počká, až se hodnota simulovaného času zvýší o náhodnou veličinu, která má normální rozdělení se střední hodnotou A a směrodatnou odchylkou B . Životní pravidlo `RELEASE X` stanoví, že transakce uvolní facility X . Trojice právě popsaných životních pravidel tedy vystihuje to, že transakce žádá danou facility o obsluhu, která má trvat po dobu určenou normálním rozdělením $N(A,B)$.

Dvojice `SEIZE` a `RELEASE` mají analogii `ENTER` (vstup) a `LEAVE` (odejdi) pro sklady. `TERMINATE` (ukončí) říká jednak to, že transakce má opustit systém, a jednak to, že končí popis životních pravidel.

Poměrně bohatý výběr možností dává životní pravidlo `GENERATE`. Uvozuje životní pravidla, je jistým obrazem toho, že transakce, která se podle nich chová, právě vychází z jistého vstupu, a dává tomuto vstupu jisté vlastnosti, a to pomocí argumentu, které za slovem `GENERATE` následují. První stanoví průměrnou dobu mezi dvěma následujícími příchody transakcí tímto generátorem; v druhém je zakódován rozptyl; třetí argument říká, kdy vypustí generátor první transakci, a čtvrtý stanoví, kolik transakcí do systému generátorem vstoupí. Po vstupu poslední transakce přestane generátor pracovat.

Kontrolní úkoly:



Zkuste v jazyku GPSS popsat nějaký jednodušší model. Např. prodejnu, kde je jeden prodavač? A zkuste popsat prodejnu, kde je více prodavačů s tím, že u každého se tvoří fronta. A zkuste popsat tutéž prodejnu, která začíná s jedním prodavačem, a pak – po jisté době – nastoupí ke svému pultu další prodavač. Dokázali byste popsat to, že ten druhý prodavač začne obsluhovat zákazníky teprve když zjistí, že fronta u toho prvního prodavače je příliš dlouhá? A dokázali byste popsat to, že když přijde druhý prodavač, přejde k němu polovina fronty od prvního prodavače? Pokud se vám některá z posledních dvou otázek bude zdát příliš obtížná, nezoufejte, neznáte ještě přece zdaleka vše!

V popisu modelu se vůbec nevyskytují fronty Q_1 až Q_6 : jazyk GPSS totiž

automaticky modeluje frontu před každou facilitou a před každým skladem, takže v běžném případě je nemusíme vůbec zavádět. Totéž platí pro výstup dat – při ukončení simulačního pokusu vystoupí automaticky tabulky a histogramy charakterizující využití jednotlivých facilit a skladů.

Úkol k zamýšlení:

V závěru minulého oddílu jste byli vybídnuti, abyste si sami uvědomili nějaký systém hromadné obsluhy, s nímž jste se setkali. Jistě jste už na něco přišli. A co takhle zkusit si to popsat v jazyku GPSS?



11.5 Popis modelu v jazyku SIMULA

Naznačme, jak by se uvedený systém hromadné obsluhy popsat v jazyku SIMULA (Simula 1989).



```
GPSS
begin integer U,počet1,počet2;
    ref(facility) Z1,Z2,Z3; ref(storage) S1,S2;
    process class generator1;
        begin loop:activate new vyrobek1; hold(10);
            if počet1 lt 1000 then go to loop
        end;
        process class generator2;
            begin loop:activate new vyrobek2;
                hold(normal(5,3,U));
                if počet2 lt 600 then go to loop
            end;
            transaction class vyrobek1;
            begin počet1:=počet1+1;
                seize(Z1); hold(normal(6,3,U));
                release(Z1);
                enter(S1); hold(normal(40,15,U));
                leave(S1);
                seize(Z2); hold(normal(5,2,U));
                release(Z2);
                enter(S2); hold(normal(50,20,U));
                leave(S2);
                seize(Z3); hold(normal(4,2,U));
                release(Z3)
            end;
            transaction class vyrobek2;
            begin počet2:=počet2+1;
                enter(S2); hold(normal(50,20,U));
                leave(S2);
                seize(Z3); hold(normal(4,2,U));
                release(Z3)
            end;
S1:-new storage(10); S2:-new storage(12);
F1:-new facility; F2:-new facility;
F3:-new facility;
activate new generator1;
activate new generator2 delay 30;
```

```
hold(100000); ...  
end;
```

Generátory jsou popsány jako zvláštní třídy, a to generator1 a generator2. Od každé z nich je pak vytvořena jedna instance. Ta první je aktivována ihned: ihned začne generovat transakce třídy vyrobek1; ta druhá je aktivována se **zpožděním** (anglicky **delay**) třiceti časových jednotek, takže začne generovat instance třídy vyrobek2 až počínaje časem 30. Na počátku se též vytvoří potřebné facility a sklady a pak se čeká 100000 časových jednotek, během nichž pracují generátory a generované transakce. Předpokládá se, že vše skončí do oněch 100000 časových jednotek. Pak se nechají vystoupit výsledky, což je zde naznačeno třemi tečkami. Zcela na začátku je prefix „GPSS“. Tím jsme naznačili, že autor programu aplikuje definice, které byly shrnuty do třídy nazvané GPSS. Jazyk SIMULA je totiž obecně použitelný (univerzální, tedy nejen simulační) programovací prostředek, takové věci jako transaction, storage, facility, seize, release, enter a depart jsou pro něj příliš specializované, takže je třeba je naprogramovat. V našem příkladu se předpokládá, že někdo v jazyku SIMULA už definoval prostředky jazyka GPSS (aniž by pro tento jazyk pracně implementoval kompilátor). Celocíselná proměnná *U* je zde rezervována pro generátor pseudonáhodných čísel.

Pojmy k zapamatování:

- systémy hromadné obsluhy
 - facilita
 - fronta
- Kendallova klasifikace
- generátor transakcí
- GPSS
- SIMULA
- sklad
 - kapacita skladu
- spotřebitel
- transakce



Kontrolní otázky:

1. Co jsou to sklady, transakce?
2. Jaký je rozdíl mezi facilitou a skladem?
3. Načrtněte jednoduché schéma obsluhového systému
4. Vysvětlete, co je to Kendallova klasifikace.



Shrnutí obsahu kapitoly

V této kapitole jsme se setkali s prvními příklady definovanými na rozmanitých systémech, které se často stávají předmětem simulace; jsou to systémy hromadné obsluhy. Pro tyto systémy existuje jistá nevelká zásoba termínů, s níž je vhodně se seznámit, protože anglické verze těchto termínů se často vyskytují v simulačních programovacích prostředcích, které



ulehčují simulaci systémů hromadné obsluhy. Anglické verze těchto termínů jsme v seznamu pojmu k zapamatování vyznačili kurzívou.

Důležité je umět zakreslit jednoduché schéma osbluhového systému ze zdroji, frontami, obsluhovými linkami a odpovídajícími vstupními a výstupními toky. Pro klasifikaci systémů hromadné obsluhy se nejčastěji používá tzv. Kendallová klasifikace, která systémy klasifikuje dle tří (resp. šesti) hlavních hledisek.

Velkou část kapitoly tvoří také modelové příklady v nástroji Witness 2007, jazyku GPSS a jazyku SIMULA. Čtenáři jsou tímto představeny různé možnosti k vytváření modelů hromadné obsluhy.

Simulují se ovšem i mnohé jiné systémy. Některé další příklady budou uvedeny v následujících kapitolách. Jsou to např. modely kompartmentových systémů, buněčné systémy, celulární automaty, Lindenmayerovy systémy, příp. epidemiologické modely.

12 Programovací prostředky pro modelování a simulaci

V této kapitole se dozvítě:

- Rozbor možností programovacích prostředků
- Simulační programovací jazyky
- Klasifikace simulačních jazyků (A, AT, T)
- Jazyky s elementárními prvky
- Kombinovaná simulace

Po jejím prostudování byste měli být schopni:

- simulační programovací jazyky,
- jejich klasifikaci a ohodnocení.

Klíčová slova této kapitoly:

Simulační jazyky, klasifikace jazyků (A, AT, T), kombinovaná simulace.

Doba potřebná ke studiu: 4 hodiny

Průvodce studiem

Simulační jazyky jsou účinné programovací prostředky, určené k ulehčení tvorby simulačních modelů na číslicových počítačích. Podstata jejich pozitivního účinku spočívá v tom, že ten, kdo je používá, nemusí popisovat přesně, co se má dít v počítači při simulačním pokusu, nýbrž popíše simulovaný systém. Takový popis je pak automaticky přeložen do programu přijatelného pro počítač.



Co je to simulační program, jsme uvedli v oddílu (2.6) druhé kapitoly. Jeho vytvoření bývá obtížné – k (číslicové) simulaci se obvykle obracíme, když máme zkoumat, optimalizovat, projektovat či upravovat složitý systém, a tak i simulační program je odpovídajícím způsobem složitý. K usnadnění – často velmi podstatnému – této programovací činnosti – slouží simulační programovací prostředky, mezi nimiž podstatným způsobem vynikají tak zvané simulační programovací jazyky. Rozborem možností simulačních programovacích prostředků, z něhož význam simulačních jazyků vyplývá, tato kapitola začíná a pokračuje popisem vlastností simulačních jazyků.



12.1 Rozbor možností

Jak už jsme poznamenali v odstavci 2.5, v dalším se budeme zabývat jen počítačovou simulací, a tak přívlastek *počítačová* budeme pro stručnost vynechávat. Je zřejmé, že simulační programy jsou velmi často složité - výpočet podle nich totiž musí být analogií nějakého složitého děje, který chceme poznat a který bývá výslednicí velmi komplexních časoprostorových a jiných vztahů.

Pro překonávání potíží se sestavováním složitých programů existuje pět způsobů, které lze v principu aplikovat i na implementaci simulačních programů:

1. sestavování programů, do nichž zabudujeme **předem připravené podprogramy**, které se chovají navenek zdánlivě jednoduše a názorně a které při tom provádějí dosti složité výpočty (tedy použití podprogramů **jako stavebních kamenů programů**);
2. použití **předem připraveného „univerzálního“ modelu**, který je řízen daty tak, že některá z nich jeho běh směrují různými cestami;
3. použití programovacích jazyků tak, že **texty** v nich sestavené a **popisující více méně čitelně požadavky** na to, co se má počítat, jsou interpretovány speciálním programem, tzv. interpretačním programem neboli interpretem;
4. použití programovacích jazyků tak, že **texty** v nich sestavené a **popisující více méně čitelně požadavky** na to, co se má počítat, jsou **překládány** do odpovídajících programů ve **strojovém kódu**;
5. použití **objektově orientovaných programovacích prostředků** jakožto základu pro definice adekvátních **problémově orientovaných** programovacích prostředků.

Hned na počátku dalšího rozboru uveďme, že bod 3 lze z následujících úvah vypustit, neboť – ačkoliv je interpretace textů historicky chápána jako samostatný způsob řešení potíží programování – jde o zvláštní případ bodu 2, což se v posledních létech stále více prosazuje: interpretace textů u programovacích jazyků interpretačním programem je zvláštním případem zpracování vstupních dat programem.

Dnes lze vyloučit z dalších úvah i bod 1, protože přípravu podprogramů vhodných pro pomoc v některých oblastech může lépe nahradit posílaní zpráv mezi jednotlivými objekty s tím, že reakce příjemců zpráv jsou adekvátní vyvolání podprogramů, mezi jejichž parametry jsou tito příjemci. Např. vyvolání podprogramu $F(X, Y)$ lze nahradit posíláním zprávy objektu X , aby provedl metodu M s parametrem Y , kde M je jen nepatrně přefor-mulovaná F . V minulosti (v šedesátých a sedmdesátých letech) bylo vyvinuto několik set „balíků podprogramů“, které podporovaly tvorbu počítačových simulačních modelů, z nichž aplikace některých přetravává někde až dodnes (např. GASP).

Univerzální či spíše univerzálnější **modely**, pomáhající podle bodu 2, se uplatňují v různých aplikačních oborech (např. v tvorbě počítačů a ve strojírenské výrobě). Dnes obvykle tyto simulátory reagují na kombinovaná alfanumerická a grafická vstupní data, jejichž rozmanitost je tak veliká, že lze těžko vyslovit nějaké společné zásady pro simulační prostředky sledující zásadu 2; shrnující popis je znemožňován i poněkud chaotickým vývojem těchto prostředků. U nás jsou z nich poněkud známy např. Taylor a Witness pro simulaci diskrétních výrobních systémů, ve světě však mají podobné prostředky nejrůznější obory lidské činnosti, např. dispečeři energetických sítí, letištění odborníci nebo pracovníci v organizaci zdravotnictví.

Zbývají tedy způsoby 4 a 5. Způsobu 4 je věnována tato kapitola, způsobu 5 pak skripta věnující se programovacímu jazyku SIMULA (Kindler 2004).

12.2 Simulační programovací jazyky

Je vhodné si uvědomit, že programovací jazyky jsou v první řadě realizovány (to jest navrhovány a implementovány) proto, aby uživateli výpočetní techniky pomohly při psaní programů. Pomoc při tom spočívá v tom, že popis pokynů, které řídí práci počítače, má být jednak pro člověka čitelný a jednak ho nemá nutit k tomu, aby zbytečně a po případě i s námahou transformoval své myšlenky.



Na těchto idejích jsou založeny tak zvané **simulační programovací jazyky**, což jsou programovací jazyky, které jsou určeny pro pomoc při sestavování simulačních programů. Než vysvětlíme jejich obecné principy, poznamenejme, že – ve shodě s terminologií ustálenou ve světě – pro ně budeme používat zkráceného názvu **simulační jazyky** (anglicky **simulation languages**), neboť neexistují žádné simulační jazyky, které by nepatřily mezi programovací jazyky.

Simulační jazyky

Když simulujeme nějaký systém, at' už existující (např. orgán konkrétního pacienta) nebo zamýšlený (např. výrobní provoz, který ještě neexistuje), předpokládáme, že takový systém už byl (a to nejednou) popsán v nějakém jazyku, který není programovací. Může to být např. odborná čeština nebo odborná angličtina, v níž si informace o objektech zájmu své práce sdělují odborníci dané profese. Z tohoto faktu vzniká přirozeně myšlenka **adaptovat jazyk dané profese jako programovací jazyk**, to jest stanovit jeho přesná syntaktická a sémantická pravidla a tím v prvé řadě odstranit nebezpečí nejasností, které přináší vždy přirozený jazyk, za druhé eliminovat některé komplikace, které slouží u přirozeného jazyka spíše ke kráse než k přesnosti vyjadřování, a za třetí umožnit strojový překlad textů v takto upraveném jazyku přímo do simulačních programů.



Ačkoliv se to vše zdá obtížné, historie ukazuje, že je to možné. Stručně řečeno, přínos simulačních jazyků spočívá v prvé řadě v tom, že uživatel takového jazyka sestavit simulační program, jehož běh by na počítači realizoval simulační model jistého systému, popíše tento systém v simulačním jazyku, a to podobně jako by ho popisoval svému kolegovi v profesi (snad jen někde se bude vyjadřovat přesněji), a popis je pak už transformován do příslušného strojového programu automaticky, v dnešní době téměř výhradně komplikací.

Je evidentní, že většinou jde o transformaci velmi složitou, takže realizace kompilátoru (překladače) je obtížná a nákladná. Nadto každý simulační jazyk by měl odpovídat nějakému profesně zaměřenému jazyku. Jelikož však profesních jazyků je mnoho, i simulačních jazyků, a tedy i jejich kompilátorů by mělo být mnoho. V mnoha profesích se však simulace tak vyplatí (a v minulosti vyplatila), že uvedené potíže neodrazují, a historie nás informuje o mnoha stech simulačních jazyků vytvořených počítačovými a softwarovými společnostmi, univerzitami i aplikujícími (např. výrobními a

projekčními) institucemi. (Jistý zlom ve vývoji simulačních jazyků, způsobený v poslední době aplikací objektově orientovaného programování, bude popsán v následující kapitole.)

Odborník, který pracuje s počítačem, má obvykle jisté schopnosti algoritmizace nebo aplikace matematických metod a je schopen jich použít i při popisu systémů, které studuje. A tak není divu, že se mezi prostředky simulačních jazyků můžeme setkat s běžnými nástroji pro algoritmizaci (s dosazováním za proměnné nebo častěji za atributy prvků simulovaného systému, s cykly, s větvením, s podmíněnými akcemi, ba i s podprogramy) a s některými matematickými nástroji, které se během doby ustálily jako nástroje popisů dějů v čase (např. s diferenčními a diferenciálními rovnicemi). Je však nutno si uvědomit, že zatímco „konvenční“ (tj. nesimulační) programovací jazyky nutí své uživatele, aby algoritmizaci prováděli do detailů a aby ji chápali jako hlavního nositele pravidel pro každý výpočet, simulační jazyky algoritmizaci nabízejí jako možnost. Když to uživatel po-važuje za vhodné, může nějaký děj, proces či rozhodnutí formulovat ve tvaru algoritmu, ale ten může „obkllopit“ popisy jiného druhu. V popisu simulovaného systému se např. některé z prvků systému mohou chovat více či méně nezávisle, přičemž o synchronizaci algoritmů prováděných několika prvků paralelně se autor popisu nemusí starat.

Příkladem může být popis modelu dopravního systému, v němž se po- hybují dopravní prostředky (např. automobily nebo vlaky). Algoritmus chování obecného dopravního prostředku (jeho reakce na signály, na okamžitý stav jeho okolí a na provádění jeho trasy) lze formulovat poměrně jednoduše, zatímco algoritmizovat synchronizaci více takových dopravních prostředků je takřka nad lidské síly; to však provede ve zkompilovaném programu už sám komplikátor. Jako další – podobně ilustrativní příklady – lze uvést boj (pozemní, tankový, letecký či námořní) mezi dvěma nepřátelskými vojenskými útvary, vývoj a dělení nádorových buněk, šíření epidemií či lesních kalamit a automatizovanou výrobu v daném podniku. Všude v těchto systémech jsou prvky, které se chovají podle pravidel, jež lze popsát jako algoritmy.

Simulačních jazyků bylo implementováno mnoho, přestože implemen-tace je pracná a nákladná. Je to dáno faktem, že prostředky každého profesního přirozeného jazyka – a tedy i každého simulačního jazyka – jsou specializovány: takový jazyk je vhodný pro popis systémů patřících do jisté **sémantické třídy**, kdežto pro ostatní systémy je více či méně nevhodný. Víme, že např. lékař by popsal fyziologii nějakého orgánu těžko v jazyku ekonoma, nebo že pracovník v oboru sociologie by měl podstatné potíže, kdyby měl popsat dejme tomu vývoj zaměstnanosti v jazyku, který používají ti, kdo konstruují lokomotivy. Existují a existovaly snahy vytvořit nějaký simulační jazyk, jehož sémantická třída by byla poměrně rozsáhlá. Výsledky těchto snah existují a jsou ve světě běžně používány. Šíře jejich sémantických tříd má však za následek, že popis každého simulovaného systému musí být proveden v detailech a „po lopatě“. Je to analogie faktu, že i přirozený jazyk široce profesně orientovaný potřebuje delší popisy objektů zájmu než jazyk orientovaný pro užší profesní skupinu, jejíž členové si

Sémantická třída



v mnohém rozumějí mezi sebou lépe než členové širší profesní skupiny. Např. v nukleární terapii nádorů je terminologie, které rozumějí odborníci v této profesi; příslušné termíny by museli vysvětlit např. jiným lékařům, s nimiž by v dané problematice přišli do styku, ale na základě obecné odborné terminologie je takové vysvětlení možné, zatímco definovat termíny nukleární nádorové terapie způsobem přijatelným dejme tomu ekonomovi nebo odborníku v hutnictví oceli je takřka nemožný (a ovšem i nepraktický) cíl.

Sémantická třída žádného z simulačních jazyků nepokrývá všechny možné dynamické systémy, takže přívlastek „univerzální“ je třeba chápát metaforicky (proto jej dáváme do uvozovek, a to i dál). Ačkoliv se zdá, že simulační jazyky v sobě spojují nevýhody jak ostatních simulačních jazyků (faktická neuniverzálnost), tak konvenčních algoritmických jazyků (nutnost podrobného popisu každého simulovaného systému), je jejich využití ve světě velmi rozšířeno. Je to jednak proto, že stále přibývají nové obory, které služeb simulace hodlají využívat, a jednak proto, že komplikace v popisech systémů, jak to simulační jazyky vyžadují, nejsou zásadní. Popis systému ve vhodném simulačním jazyku je stále mnohem jednoduší a čitelnější než v konvenčních programovacích jazycích.

12.3 Základní klasifikace simulačních jazyků

Už v první kapitole jsme se zmínili, že v systémech existují obecně transakce a permanentní prvky čili aktivity. Systémy, které obsahují prvky obou těchto kategorií, budeme nazývat **systémy typu AT** (zkratka Aktivity-Transakce) nebo zkráceně **AT-systémy**. Lze si představit systémy tak proměnné v čase, že v nich žádné aktivity nejsou, tj. že všechny jejich prvky chápeme jako transakce. Tyto systémy budeme nazývat **systémy typu T** čili **T-systémy**. Existuje však i opačný extrém, **systémy typu A** čili **A-systémy**, v nichž nejsou žádné transakce. V takových systémech zanedbáváme v podstatě všechny strukturní změny a celý děj, který v nich v čase probíhá, abstrahujeme do nějakých změn hodnot atributů (většinou jde o atributy numerické, resp. booleovské).

Nechť L je simulační jazyk a C jeho sémantická třída. Když C obsahuje pouze A-systémy, řekneme, že L je **jazykem základního typu A**. Když C obsahuje pouze T-systémy, řekneme, že L je **jazykem základního typu T**. V ostatních případech řekneme, že L je **jazykem základního typu AT**.

Upozorňujeme, že sémantická třída jazyka základního typu AT nemusí být omezena na AT-systémy. Existují sice jazyky základního typu AT, v nichž lze popsat pouze systémy obsahující aspoň jednu aktivitu a aspoň jednu transakci, avšak obecně platí, že sémantická třída jazyků základního typu AT může obsahovat i systémy typu A (což jsou v mnoha případech degenerované případy systémů typu AT) a – v případech mnohem méně častých – i systémy typu T.



Systémy typu AT

Systémy typu T

Systémy typu A

Jazyky základního typu A, T, AT



Úkoly k zamýšlení:

Zkuste najít ve svých vzpomínkách nějaký A-systém a nějaký AT-systém. Jak byste jej ve své vlastní systémové abstrakci přeměnili na T-systém? A znáte nějaký jiný T-systém?

12.4 Jazyky základního typu A

Sémantické třídy těchto jazyků jsou dosti omezeny, a tak lze tyto jazyky charakterizovat velmi prostě: až na několik (v zásadě bezvýznamných) výjimek jde o následující tři skupiny.

Jazyky pro spojitou simulaci

Tak zvané **jazyky pro spojitou simulaci** (anglicky **continuous systems simulation languages**) jsou simulační jazyky, v jejichž sémantických třídách jsou spojité systémy, které lze popsát pomocí obyčejných diferenciálních rovnic, v nichž se vyskytují derivace atributů jednotlivých prvků podle času. Prostředky těchto jazyků vycházejí obvykle z ustálené matematické symboliky: rovnost časové derivace atributu x pravé straně $f(x, \dots)$ může uživatel mnoha takových jazyků – zejména amerických – přepsat na rovnost hodnoty atributu x časovému integrálu pravé strany, přičemž symbol integrálu se nahrazuje v různých jazycích různými alfabetickými výrazy jako INT, INTEG, INTEGRAL apod. Místo $x' = f(x, \dots)$ se tedy musí uvést $x = \text{INTEG}(f(x), \dots)$; poznamenejme, že dt se přitom vynechává. Starší (avšak zejména v USA stále používané) jazyky pro spojitou simulaci vedou jejich uživatele k tomu, aby simulovaný systém popsali jako konkrétní zapojení idealizovaného analogového počítače, tj. systému složeného z analogových sčítaček, násobiček, integrátorů a dalších prvků, jejichž nabídka fakticky charakterizuje úroveň takového jazyka.

Jazyky pro simulaci technického vybavení počítačů

Tak zvané **jazyky pro simulaci technického vybavení počítačů** (anglicky **hardware simulation languages**, někdy – ne zcela přesně – nazývané **hardware description languages**, tj. jazyky pro popis technického vybavení počítačů) jsou simulační jazyky, které umožňují svým uživatelům popsát strukturu počítače nebo jeho části v termínech, jež jsou obvyklé mezi počítačovými techniky. Tyto jazyky lze klasifikovat do tří podskupin, a to podle toho, na které úrovni umožňují popis (a tedy i simulaci) počítače nebo jeho části:

1. **Jazyky na úrovni obvodů.** Uživatel popíše procesor nebo jeho část tak, jak by ji popsal specialistovi v elektrotechnice, tj. jako síť tranzistorů, kondenzátorů, odporů a dalších prvků. Reakce počítače spočívá ve spojité simulaci elektronických dějů v takovém systému. Je evidentní, že jazyky tohoto typu lze s výhodou aplikovat i na simulaci elektronických obvodů, které nemají s výpočetní technikou nic společného; přesto se však ve světové literatuře zahrnují - byť nepřesně - mezi jazyky pro simulaci technického vybavení počítačů.
2. **Jazyky na úrovni logických hradel.** Uživatel popíše procesor nebo jeho část jako síť složenou z „logických prvků“, v níž se pohybují signály (tedy logická nula a logická jednička). Logické prvky reagují na své vstupní signály způsobem, který lze popsát pomocí logických operací známých např. z matematické logiky (logické **and** čili a, **or** čili nebo, **not** čili ne apod.). Logické prvky někdy mohou své

výstupní hodnoty v čase zpozdit. Reakce počítače na takový popis je diskrétní simulace děje v popsané síti.

3. **Jazyky na úrovni registrů a mikroprogramování**, nazývané v angličtině **register transfer languages**. Uživateli dávají k dispozici knihovnu standardních prvků odpovídající takovým složkám počítače, jako jsou registry různých druhů a další paměťová místa, sčítáčky, střádače, násobičky atd. Reakce počítače na takový popis je diskrétní simulace dějů v popsaném systému.

Další je skupina jazyků zaměřených na simulaci transportních procesů pomocí tzv. systémové dynamiky. Ta nemá speciální název ve světě běžně používaný, protože takových jazyků je velmi málo. Mají však důležitou roli ve spojitém modelování ekonomických toků (toků financí nebo materiálu) a – přibližně od konce 80. let – i procesů v životním prostředí. Jejich uživatel popíše studovaný systém jako soustavu „nádob“ spojených „kanály“, jimiž se mezi nádobami pohybuje „látka“; ta může odpovídat chemické látce nebo množství jedinců v systémech životního prostředí nebo financím či množství obchodovaného produktu v ekonomii. Rychlosti v kanálech mohou být vyjádřeny pomocí výrazů, v nichž se vyskytují objemy „nádob“, takže tyto rychlosti a tím i objemy se mohou v čase měnit dosti složitým způsobem. Uvedená metoda popisu byla koncem 50. let navržena J. Forresterem, který ji s úspěchem nabídl ekonomům, a ti občas piší o forresterovských modelech a jazycích. Podle nejznámějšího z jazyků této skupiny, který nese název DYNAMO, se občas setkáme i s termínem „jazyky typu DYNAMO“. Čtenář kapitoly 10 jistě pocítil velkou příbuznost těchto jazyků s jazykem COSMO. Je to tak, tento jazyk do zde popisované skupiny lze zařadit, avšak má proti jejím ostatním členům něco navíc, a to tracer (prostředky pro manipulaci s něčím podobným DYNAMO ani ostatní jazyky této skupiny nemají).

Pomocí uvedených jazyků základního typu A bylo možno nejprve simulovat systémy o desítkách rovnic, resp. prvků, avšak během doby – jak se zvyšovala rychlosť a paměťová kapacita počítačů – se rozsah simulovaných systémů zvětšoval. Možnosti dnešních počítačů nevyulučují simulovat systémy, které mají mnoho set prvků, avšak v takovém případě se narází na hranice lidské psychiky. Jedinec není schopen popsat bez chyb tak velký systém prvek za prvkem. A proto takřka všechny moderní jazyky základního typu A umožňují zavádět tzv. **makra**, což jsou jakési podsystémy, které lze kopírovat a které uživatel může definovat (nebo převzít z knihovny maker) a použít je stejně jako ostatní – jednodušší – bloky, resp. rovnice. Např. v jazyku pro spojitu simulaci může uživatel definovat makro „chemický reaktor“ jako systém složený z mnoha (analogových) sčítáček, integrátorů a dalších prvků, podobně může definovat makra „výměník tepla“, „rektifikační kolona“ a další a pak formulovat, že jím popsaný simulovaný systém chemické výroby je síť složená z takových a takových chemických reaktorů, výměníků tepla, rektifikačních kolon a dalších prvků, tak a tak propojených. Kompilátor pak „nakopíruje“ do paměti pro každé makro jeho vnitřní strukturu podle toho, jak bylo dané makro definováno, takže např. každý chemický reaktor má sčítáčky, integrátory a další prvky, které mu dle definice přísluší.

Makra

Pro výše zavedené názvy skupin jazyků základního typu A jsme vycházeli z toho, že jsou mezinárodně přijaté. Musíme však upozornit, že ani jeden z názvů není zcela výstižný. Tak např. jazyky pro simulaci elektronických obvodů by měly logicky patřit mezi jazyky pro spojitu simulaci, ale odborná veřejnost to tak nebere. Nadto téměř všechny dnes používané jazyky pro spojitu simulaci mohou připouštět nespojitosti v chování simulovalaného systému; příkladem je blok nazývaný *signum* (znaménko), který má jeden vstup a na svém výstupu dá jednu z hodnot 1 a -1, a to podle toho, zda je číslo na vstupu nezáporné nebo záporné. Zatímco výstup ze sčítáčky se v čase mění spojite, pokud se takto mění i hodnoty na jejím vstupu (a výstup z integrátoru se mění spojite dokonce i tehdy, když integrovaná hodnota na jeho vstupu se mění skokem), u prvku *signum* se změní hodnota na jeho výstupu skokem, když se hodnoty na jeho vstupu mění spojite od kladných na záporné nebo naopak.

Existují spojité systémy, které nelze popsat pomocí obyčejných diferenciálních rovnic. Platí to např. pro tak zvané **systémy s rozdělenými parametry**, při jejichž popisu musíme použít parciálních diferenciálních rovnic.

Systémy s rozdělenými parametry

Tak vzniká otázka, kam zařadit simulační jazyk, který umožnuje popsat systémy s rozdělenými parametry. Světová odborná veřejnost se na jednotné odpovědi dosud neshodla, avšak poznamenejme hned, že to zatím nevadí. S numerickými metodami, které by bylo možno algoritmizovat pro řešení mnoha parciálních diferenciálních rovnic, jsou tak velké potíže, že – zatímco jazyků na úrovni obvodů je několik set – lze simulační jazyky orientované na systémy s rozdělenými parametry spočítat na prstech; v klasifikaci se o nich už dále nebudeme šířit.

Úkol k zamýšlení:



S jakými systémy typu A jste se setkali?

Na závěr je nutno ještě upozornit, že logicky by mezi jazyky pro simulaci technického vybavení počítačů měly patřit i jazyky pro simulaci počítačových sítí a složitějších systémů složených z procesorů. Takové jazyky sice existují, ale to už nejsou jazyky základního typu A, neboť v jejich semantické třídě existují např. počítačové sítě, které si předávají úlohy a protokoly, což jsou transakce: mají totiž své atributy, které během jejich existence informují o jejich charakteru a o tom, jak budou např. interagovat s tím či oním prvkem sítě.

12.5 Jazyky základního typu AT

Simulační program

Vyjděme z příkladu počítačových sítí, kterým byl ukončen předcházející odstavec. Počítačovou síť zpravidla simulujeme tak, že předpokládáme pevnou strukturu jejich počítačů, terminálů a spojů mezi nimi, avšak připouštíme, že v síti těchto permanentních prvků se „pohybují“ úlohy, že tyto úlohy do sítě vstupují a po čase z ní mizejí. Jsou to tedy transakce. Počítačová síť je tedy A-systém.

Místo počítačové sítě si můžeme představit železniční síť, síť pouliční dopravy ve městě, letištní plochu, velkou nádražní nebo letištní budovu,

výrobní nebo montážní systém, vše má permanentní prvky a transakce. Jde o tak zvané **systémy hromadné obsluhy**, tj. systémy, v nichž se tvoří fronty – většinou uspořádané množiny transakcí. Připomínáme, že se systémy hromadné obsluhy jsme se setkali už v kapitole 9.

Systémy
obsluhy

hromadn

Když chceme nějaký systém simuloval, musíme ho více méně přesně v simulačním programu popsat; musíme si tedy vždy uvědomit jisté vlastnosti systémů, které zamýslíme simuloval. Odborníci si brzy (koncem 50. let) povšimli, že pro systémy hromadné obsluhy hrají podstatnou roli děje spojené s jednotlivými transakcemi nebo aktivitami. To, co se v systému hromadné obsluhy děje, je výslednicí toho, jak se v systému „pohybují“ transakce nebo jak v něm aktivity „pohybují“ transakcemi.

Příklad:



Příkladem může být vlak v železniční síti. Lze na něj jako na transakci pohlížet tak, že má svou trasu (seznam míst, kterými má projet) a podle ní provádí cyklus odpovídající přesunům z jednoho místa do místa následujícího. Každý krok tohoto cyklu obsahuje vlastní přesun, stání ve stanicích a případné stání před semafory. To, zda je nutno zastavit před semaforem, je dán testem, zda je na semaforu „zelená“, tj. na to, jaká je hodnota jistého atributu aktivity, jíž semafor je. (Poznamenejme, že v tomto příkladě lze takový atribut chápat jako booleovský, nazvaný dejme tomu *volno*, jehož hodnota je buď *ano* (vlak může jet) nebo *ne* (vlak musí čekat).) U každého vlaku pak můžeme pomocí simulace určit, jak mnoho času prostojí před semafory. Simulovaný systém tedy popíšeme tak, že vlak má numerický atribut *čas strávený čekáním před semafory*, k jehož hodnotě se na konci každého čekání před semaforem přičte délka tohoto čekání.

Na základě tohoto jednoduchého příkladu poznáváme, že typický „děj“ vlaku lze popsat algoritmicky podobně jako např. podprogram; i v ději vlaku jsou dosazení (za atributy), větvení, resp. podmíněné příkazy (test na „zelenou“ semaforu a následující čekání nebo vypuštění tohoto čekání) a – jak už jsme naznačili – cykly. Je evidentní, že v jiných případech simulovaných systémů mohou být algoritmy dějů transakcí mnohem složitější a tvůrce simulačních programů přitom s chutí použije i jiné prostředky algoritmizace jako např. podprogramy.

Zústaňme ještě u příkladu s vlaky. Děj v železničním systému můžeme popsat také jiným způsobem. Spojíme jeho akce s aktivitami (stanicemi, semafory, kolejovými úseky atd.), jako kdyby je tyto prvky aktivně vykonávaly a „pohrávaly“ si přitom s transakcemi – vlaky. Vlaky jako by byly pasivními prvky, bez vlastního děje: ani hodnoty svých atributů si nemění samy, ty jim mění aktivity, když si s nimi „pohrávají“.

Tento objev lze rozšířit i na systémy, kde nevznikají fronty. V kapitole 11 jsme se seznámili s faktem, že život buňky, tj. její přecházení z jedné fáze do druhé, přičemž se v některé může rozdělit, lze popsat algoritmem. Buňky jsou transakce a např. jejich stav (nebo mnohem složitější biolo-

gické entity související nejen s buněčným stavem, ale i s jejím umístěním apod.) jsou aktivity.



Kontrolní úkol:

Jistě jste se už setkali s nějakým AT-systémem. Zkuste jeho pravidla nějak popsat vzhledem k jeho transakcím nebo vzhledem k jeho aktivitám.

Odborníci si výše uvedených jevů brzy všimli a využili toho, co zpozorovali, k návrhu simulačních jazyků základního typu AT. Vznikly dvě kategorie těchto jazyků, které budeme nazývat **jazyky typu AT** (nebo stručně **AT-jazyky**) a **jazyky typu TA** (stručně **TA-jazyky**). Upozorňujeme, že slovo „základní“ zde tedy vynecháváme.

Jazyky typu AT

Jazyky typu AT umožňují svým uživatelům popsát systém typu AT tak, že se popíše „skelet“ aktivit a třídy transakcí. Pro každou třídu transakcí se uvede, jaké mají její transakce atributy, a dále „algoritmus životních pravidel“, jímž se každá její transakce řídí. **Jazyky typu TA** umožňují svým uživatelům popsát systém typu AT tak, že popíší aktivity spolu s „algoritmy jejich životních pravidel“, ovšem s tím, že mezi takovými aktivitami mohou být „generátory transakcí“, které vytvářejí transakce a posílají je k jiným aktivitám.

Prvky aktivní a pasivní

Prvky, s nimiž jsou spojena životní pravidla, lze nazvat **aktivními prvky** a ostatní **prvky pasivními**. Jak je patrno z právě zavedených názvů, využíváme při identifikaci typu (nikoliv základního typu!) simulačního jazyka jejího posledního místa k určení kategorie aktivních prvků; např. AT říká, že děj nesou transakce. Tuto konvenci budeme aplikovat i později.

Popis simulovaného systému v některém z jazyků typu AT či TA je komplítorem jazyka zpracován tak, jak to odpovídá skutečné existenci v čase: algoritmy prováděné jednotlivými prvky, tj. životní pravidla, jsou „synchronizovány“ ve společném čase (řečeno neoborně, jejich provádění je promícháno). Jako ilustraci si můžeme představit to, že když na nějakém úseku železniční sítě se pohybuje vlak V , na jiném úseku se může rozjet jiný vlak, který může vlaku V zablokovat anebo naopak odblokovat vjezd do stanice či příjezd k semaforu. Tato automatická synchronizace je jedním z největších přínosů, které jazyky typu AT a jazyky typu TA poskytují.

Vliv společného času, v němž by měly prvky existovat, a s ním spojená „synchronizace“ tvoří důležitou odlišnost životních pravidel od běžných algoritmů. Když např. popisujeme životní pravidla vlaku, musíme v nich uvést, že některé jejich kroky zaberou nějaký čas. Kdyby čas neměl význam, nebylo by možné ani synchronizovat to, jak jsou různými prvky simulovaného systému prováděna jejich životní pravidla. Jazyky typu AT i jazyky typu TA řeší problém nenulového trvání jednotlivých kroků životních pravidel pomocí jakýchsi přerušení těchto pravidel: např. tvrzení, že vlak se pohybuje z místa A do místa B spojite a nějakou dobu mu to trvá, popíše uživatel těchto jazyků obvykle tak, že vlak je v místě A , pak nějakou dobu (provádění svých životních pravidel přeruší) a pak se najednou octne

v místě *B*. Krok, kterým vlak přeruší provádění svých životních pravidel, se nazývá **plánovací příkaz** (anglicky **scheduling statement**).

Rozeznávají se dva hlavní druhy plánovacích příkazů: **imperativní** čili rozkazovací příkaz (angl. **imperative statement**), který bychom mohli vyjádřit slovy „čekej po dobu tak a tak dlouhou“, a **interrogativní** čili tázací příkaz (angl. **interrogative statement**), který lze přiblížit slovy „čekej, až bude splněna daná podmínka“. Trvání přesunu vlaku mezi dvěma místy se evidentně vyjádří imperativním příkazem („čekej po dobu nutnou k přesunu“), kdežto čekání na uvolnění dráhy změnou signálu vysílaného semaforu vyjádříme interrogativním příkazem („čekej, až se změní barva na semaforu“).

S plánovacími příkazy a dokonce i s poukázáním na jejich význam už jsme se setkali v oddíle 11.3 a s konkrétními plánovacími příkazy jsme se setkali již před tím, v oddílech 9.2 (*advance, seize, enter*), 9.3 (*hold, seize, enter*) a 11.2 (*hold*). Příkazy *hold* a *advance* jsou imperativní, příkazy *seize* a *release* jsou interrogativní; např. *seize(x)* nese ve svém parametru *x* podmínu pokračování tvaru „až bude možno interagovat s facilitou *x*“.

12.6 Jazyky základního typu T

V exaktních oborech je běžné, že něco, co neexistuje, se chápe jako cosi, co by mohlo existovat, ale „má to nějaký nulový parametr“. Místo abychom řekli, že měření nebo výpočet byl bez chyby, řekneme, že jeho chyba je nulová. Místo toho, abychom řekli, že nějaký fyzický objekt není hmotný, resp. nenese elektrický náboj, řekneme, že hmotnost resp. elektrický náboj objektu jsou nulové. Podobný přístup lze aplikovat i pro aktivity: lze je chápout jako transakce, které „nevyužívají“ možnost měnit svou vzájemnou konfiguraci, vstupovat do systému po začátku jeho existence a opouštět jej před koncem jeho existence. Jelikož systém je abstrakce definovaná na realitě, je takový přístup oprávněný: v takové abstrakci prostě abstrahujeme od toho, že víme o vzájemné neměnnosti některých prvků daného systému.

Je vhodné si uvědomit, že myšlenkovou cestou, kterou jsme právě naznačili, se dostáváme velmi jednoduše ke konkrétním T-systémům. Vidíme, že pro T-systémy není nutno studovat a modelovat nějaké těžko zvládnutelné objekty, v nichž „se vše mění“.

Uvedené pojetí bylo realizováno v roce 1964 v simulačním jazyku SIMULA, který byl později nazýván SIMULA I (Dahl a Nygaard 1967). Nebyl to ještě známý objektově orientovaný jazyk (nazývaný v prvních létech své existence SIMULA 67), jehož principy byly sděleny světové odborné veřejnosti až roku 1967, nýbrž vskutku jazyk zaměřený na simulaci (diskrétních) systémů. Jeho autoři jej charakterizovali jako jazyk pro programování a popis systémů s diskrétními událostmi (language for programming and description of discrete event systems). Jeho uživatel byl vskutku veden k tomu, aby v případě, že by chtěl popsat nějaký systém typu AT, chápal aktivity jako degenerované transakce, u nichž není využito možností měnit jejich vzájemnou konfiguraci, vstupovat do systému po

Plánovací příkaz:

imperativní,

interrogativní

začátku jeho existence, opouštět je před koncem jeho existence a dokonce mít životní pravidla.

Simulačních jazyků s vlastním komplátorem, které jsou typu T, je velmi málo. Přibližují se k nim však simulační jazyky definované na bázi objektově orientovaného programování, tj. simulační jazyky bez vlastních komplátoreů.. K tomuto fenoménu se však vrátíme ještě v následující kapitole.

12.7 Jazyky s elementárními prvky

Jak už jsme poznamenali, mají životní pravidla mnoho analogií s algoritmy. Tento fakt podněcuje a už dávno podněcoval tři ideje. Ta první, mohli bychom říci skromnější, spočívá ve snaze použít k popisu životních pravidel zvyklostí (včetně konkrétních syntaktických pravidel) známých a ustálených v konvenčních algoritmických jazycích. Uživatel simulačního jazyka by se nemusel učit pravidla, jak zapisovat dosazení, cykly, větvení a další vyjadřovací algoritmické prostředky, nýbrž by aplikoval své znalosti a dovednosti z jiného jazyka (v úvahu přicházely postupně FORTRAN, ALGOL, BASIC, C a JAVA). Druhá, méně skromná idea spočívá v tom, že by se životní pravidla (a vlastně celé popisy simulovaného systému, tedy celé texty v simulačním jazyku) mohly automaticky překládat do textů v nějakém konvenčním algoritmickém jazyku: algoritmické prostředky by se prostě okopírovaly a komplátor simulačního jazyka by se podstatně zjednodušil. A třetí, nejnáročnější idea spočívá v tom, že by vůbec nemusel existovat nějaký překladač či komplátor, kdyby se algoritmické struktury nějakého rozšířeného programovacího jazyka doplnily vhodnými podprogramy pro generování a likvidování transakcí a pro plánovací příkazy.

Realizace první ideje, pokud je izolovaná od druhé a třetí, nevyžaduje nějaké zvláštní triky při implementaci simulačního jazyka, avšak nemá sama ani nějaký podstatný dopad. Např. algoritmické prostředky jazyků GPSS a GERT, o nichž jsme se zmínili v závěru předcházejícího odstavce, jsou daleko od všech zvyklostí známých z konvenčních algoritmických jazyků.

Pokud chceme realizovat druhou, resp. třetí, ideu, je otázka, jak to udělat. Konkrétně vzato, u druhé ideje je třeba odpovědět na otázku, jakou formu bude mít text vzniklý překladem životních pravidel třídy prvků. A u třetí ideje musíme vědět, čemu by měly odpovídat v oblasti konvenčních algoritmických jazyků životní pravidla.

Odpověď se zdá být prostá: životní pravidla formulovaná pro třídu prvků jsou prováděna postupně vsemi jejími instancemi, a tak by bylo vhodné chápat životní pravidla jako podprogram (nebo je jako celek do programu překládat): podprogram je přece také cosi, co se formuluje jednou, ale může to být vykonáváno mnohokrát.

Naneštěstí taková odpověď je nepřesná a nepoužitelná. Potíž je v „přepínání“ mezi životními pravidly, jak jsme se o tom zmínili v souvislosti s plánovacími příkazy: něco podobného se mezi podprogramy totiž neděje. Ukázalo se, že jediný přístup, jak životní pravidla více či méně převést na

texty v konvenčních algoritmických jazycích, spočívá v rozdělení životních pravidel na nepřerušitelné, to jest okamžité akce a ty formulovat jako podprogramy. Nechceme zde zabíhat do detailů, a tak uvedeme jen několik postřehů.

Životní pravidla mají tvar $A_1, P_1, A_2, P_2, \dots, A_n, P_n, A_{n+1}$, kde A_i jsou úseky, které lze vyjádřit pomocí prostředků konvenčních algoritmických jazyků, a P_i jsou plánovací příkazy, dejme tomu tvaru „čekej po dobu T_i “ nebo „čekej, až bude splněna podmínka B_i “. Definujeme v systému $n + 1$ tříd $E_1, E_2, \dots, E_n, E_{n+1}$ jakýchsi fiktivních prvků – budeme jim říkat **elementární prvky** – které jsme původně v systému nepozorovali. Každý z těchto elementárních prvků má jeden referenční atribut R a s třídou E_i jsou spojena pravidla A_i , za nimiž následuje ještě doplněk D_i , který má tvar „generuj prvek třídy E_{i+1} , okopíruj na jeho atribut R současnou hodnotu atributu R a aktivuj tento prvek za čas T_i (resp. až bude splněna podmínka B_i). Místo abychom k dané třídě formulovali životní pravidla ve tvaru

$$A_1, P_1, A_2, P_2, \dots, A_n, P_n, A_{n+1},$$

necháme ji pasivní (bez životních pravidel). Když je však generován prvek X uvažované třídy, spolu s ním necháme generovat i prvek třídy E_1 a za jeho atribut R mu dosadíme X . Když si v duchu probereme, co se bude dít, zjistíme, že prvek X bude postupně „ovládán“ prvky s životními pravidly $A_1, A_2, \dots, A_n, A_{n+1}$, totiž prvky, které se budou postupně vytvářet, předávat si „ovládání“ prvku X pomocí atributu R a čekat při krocích tohoto ovládání zcela stejně, jako by tomu bylo dle původně zamýšlených životních pravidel.

Životní pravidla elementárních prvků nejsou přerušována plánovacími příkazy, a tak splňují to nejdůležitější: elementární prvky lze bez komplikací převést na podprogramy, ba je možno vše zařídit tak, že přímo – tedy bez překladu – se elementární prvky mohou formulovat jako podprogramy. Lze tedy realizovat druhou, ba dokonce i třetí ideu. Avšak zavádění elementárních, tedy fiktivních, prvků je nepřirozené a „rozbíjení“ původně zamýšlených celistvých životních pravidel mezi elementární prvky je komplikované, stejně jako explicitní naprogramování toho, aby si tyto prvky v daných časových relacích „předávaly vládu“ nad jedním a týmž prvkem simulovaného systému.

Výhody – jednoduchost kompilátoru nebo dokonce jeho úplné odstranění – vedly v 60. letech k jisté oblibě jazyků s elementárními prvky. Patřily sem populární simulační jazyky SIMSCRIPT I a II (vyžadující kompilátory) a GASP. Postupem doby však převládl v lidské civilizaci nesouhlas s obtížemi při programování, takže aplikace takových jazyků takřka vymizela. V 90. letech nastala jistá renesance, a to ve vztahu k objektově orientovaném programování (viz následující kapitola).

Jazyky s elementárními aktivitami patří vskutku minulosti a dnes stojí za to uvažovat o případech, kdy elementární prvky jsou transakce. Když je symbolizujeme výrazem T_E , můžeme říci, že právě zmíněné jazyky jsou

Elementární prvky

Událost

Kvaziparalelní systém
Kvaziparalelní
programování

typu ATT_E a patří mezi jazyky základního typu AT. V americké literatuře se elementární prvky nazývají **events** (česky **události**), avšak tento termín je – i v angličtině – používán i v jiných souvislostech (mimo jiné i v oboru jazyků typu TA a typu AT – tedy jazyků bez elementárních prvků – pro nepřerušené úseky výpočtu, tedy pro úseky, které jsme výše značili jako A_1, A_2, \dots, A_{n+1}), a tak ho nedoporučujeme v uvedeném významu používat. Odstraní se tím mnoho zbytečných nedorozumění.

Souhrn prostředků pro deterministické přepínání výpočtu mezi několika algoritmy se nazývá **kvaziparalelní systém** a programování s použitím kvaziparalelních systémů se nazývá **kvaziparalelním programováním**. Zdůrazňujeme, že jde o deterministické přepínání, tj. takové, které nezávisí na okamžitém stavu počítače. (Nedeterministické přepínání známe např. z multiprogramování a multitaskingu, jak jsou zabudovány v některých operačních systémech.) Jazyky typu AT a typu TA tedy mají zabudován kvaziparalelní systém, zatímco použití jazyků s elementárními prvky je charakteristické tam, kde stavíme na základě (konvenčního) programovacího jazyka, který kvaziparalelní programování neumožňuje. Běžně oblíbené programovací jazyky (FORTRAN, Pascal, Basic, LISP, SmallTalk, JAVA, C ani C++) kvaziparalelní programování neumožňují.

Detailly týkající se klasifikace simulačních jazyků a příklady některých konkrétních jazyků lze nalézt v monografii (Kindler 1980).

12.8 Poznámka ke kombinované simulaci

V předchozí kapitole jsme mezi jednotlivými druhy simulačních jazyků uváděli pouze jazyky pro spojitou simulaci. Nezmínili jsme se o jazycích pro kombinovanou simulaci, abychom klasifikaci z hlediska vztahu k aktivitám a transakcím příliš nezatemnili.

V kapitole 1 jsme naznačili, že kombinovaná simulace se chápe jako simulace systému, který má podstatné vlastnosti jak systémů spojitéch, tak i systémů diskrétních. Kombinovaná simulace není tedy jen nějaké malé „poškození“ spojité simulace nějakými nespojitostmi nebo nějaký spojity fenomén v jinak diskrétním systému (např. rovnoměrný či rovnoměrně zrychlený pohyb transakcí, při jehož modelování by stačilo počítat prostoročasové souřadnice výchozích a výsledných – tedy diskrétních – stavů řešením rovnic, které se každý může naučit v prvních týdnech středoškolských hodin fyziky).

Z těchto požadavků, jež klade na kombinovanou simulaci světová odborná veřejnost, vyplývají i požadavky na odpovídající programovací prostředky: jazyky pro kombinovanou simulaci musí povolit jednak popis systémů s transakcemi, jednak popis spojitéch změn určených nejméně obyčejnými diferenciálními rovnicemi. Z prvního požadavku plyne, že jazyky pro spojitou simulaci jsou základního typu AT nebo základního typu T. Druhá vlastnost souvisí s otázkou, která existuje už v souvislosti se spojitou simulací, jak široce se vlastně má pojem spojitosti v simulaci chápát. Odpověď na tuto otázkou souvisí s odpovědí, kterou dává právě skutečnost prostředků pro spojitou simulaci, kde se spojitost chápá jako chování podle



obyčejných diferenciálních rovnic s derivacemi podle času. I v kombinované simulaci jsou takové rovnice postačujícím důvodem pro přijetí spojitosti. Při psaní těchto učebních textů byl znám jediný jazyk pro kombinovanou simulaci (GASP V), který připouští, aby jeho uživatel popisoval složky simulovaného systému i s pomocí parciálních diferenciálních rovnic, zatímco všechny ostatní jazyky pro kombinovanou simulaci jsou omezeny jen na diferenciální rovnice obyčejné.

Řečeno prozatím velmi stručně, v kombinovaném systému tedy existují prvky, jejichž některé atributy se mohou měnit (alespoň někdy) spojite, a dále v něm existují diskrétní události, včetně vzniku a zániku transakcí. Avšak to není všechno. Vyskytuje-li se v nějakém systému spojité a diskrétní změny, lze právem očekávat, že se v něm tyto změny dostanou i do nějaké vzájemné závislosti, a tak jazyky pro kombinovanou simulaci musí být vybaveny prostředky pro popis těchto závislostí. Jde konkrétně o dva druhy závislostí.

- Diskrétní událost způsobí diskrétní změnu hodnoty atributu, která se jinak mění spojite.
- Spojite se měnící atribut dosáhne jistého stavu, což způsobí diskrétní událost, která se může projevit na zcela jiných vlastnostech systému než na daném atributu. Taková událost se nazývá **stavovou událostí** (anglicky **state event**). Nevylučuje se, že kritická událost může aktivovat čekající proces nebo naopak pasivovat jiný proces včetně toho, jehož atribut k této události svým spojitým průběhem vedl. Nevylučuje se ani to, že kritická událost způsobí vznik nových transakcí nebo naopak likvidaci jiných včetně té, jejíž atribut tuto událost způsobil.

Stavová událost

Pro diskrétní změnu hodnoty atributu, který se jinak mění spojite (případ 1), není třeba vymýšlet nějaký nový vyjadřovací prostředek. Jde o zcela obvyklé dosazení, stejně vyjádřené jako dosazení nové hodnoty za atribut, který se spojite nemění. V implementaci jazyka to ovšem může znamenat jisté komplikace, ale o ty se uživatel nemusí starat (a proto si jich v těchto učebních textech nevšímáme).

Pro stavovou událost se nabízí několik možností. Jazyky s elementárními prvky (viz odstavec 12.7) mohou snadno nabídnout uživateli, aby stavovou událost popsali jako elementární transakci, pro kterou platí jistá podmínka, kdy nastane, a to podmínka, že daný atribut dosáhne kritické hodnoty. Ostatní simulační jazyky (tedy konkrétně jazyky základního typu AT či T) řeší celou věc tak, že jeden nebo více prvků, které provádějí svá životní pravidla, narazí na jistý speciální druh interrogativního plánovacího příkazu (viz odstavec 12.5), totiž příkazu „čekej, až daný atribut dosáhne jisté (totiž té kritické) hodnoty“.

Když se nad stavovými událostmi hlouběji zamyslíme, dojdeme k názoru, že stavová událost by mohla nastat obecně v okamžiku, v němž hodnoty nějakých atributů vyhoví nějaké podmínce, čili když splní nějaký booleovský výraz. Takový výraz přece nemusí být jen ta nejjednodušší relace,

tedy relace tvaru $a = k$, kde a je uvažovaný atribut a k je konstanta, nýbrž např. $a_1 = a_2$ (kritická událost nastane, když se hodnoty dvou atributů budou sobě rovnat), nebo $a_1 = k_1 * a_2 + k_2$, kde a_1 a a_2 jsou nějaké atributy (dokonce nemusí být oba současně spojité) a k_1 a k_2 nějaké konstanty (nebo dokonce také atributy).

Na dotazy, které při takovém rozboru v mysli každého mohou vzniknout, odpovídají jednoznačně fakta o jazycích pro kombinovanou simulaci. Zdá se, že se všechny dnes existující jazyky pro kombinovanou simulaci omezují právě jen na ten jednoduchý případ dosažení kritické hodnoty, případně povolují zadat i směr, odkud je tato hodnota dosažena (událost nastane, je-li hodnota dosažena zdola, resp. shora, resp. na směru dosažení nezáleží). Možnosti složitější formulace podmínky pro vznik stavové události se dnes uživatelům nenabízejí a ti si musí s takovými případy poradit sami (v praxi ovšem jsou podobné případy spíše výjimečné).

Na závěr uvedeme klasifikaci jazyků pro kombinovanou simulaci, která vyjadřuje jakousi schopnost jazyka popsat více nebo méně. Budeme mluvit o třech kombinovaných typech jazyka, a to K_1 , K_2 a K_3 .

Jazyky kombinovaného typu K_1 povolují popsat systémy složené z jedné aktivity C a z „diskrétního podsystému“ D . C má atributy, jejichž změny se mohou popsat pomocí diferenciálních rovnic, kdežto v D nic podobného neexistuje, to je vskutku diskrétní systém. Prvky v D však mohou způsobit diskrétní změny atributů v C a atributy v C mohou vést ke stavové události v D . Ve Spojených státech populární jazyky GASP IV a jejich odvozeniny jsou tohoto typu, stejně jako výše zmíněný GASP V.

Jazyky kombinovaného typu K_2 povolují, aby jakákoli transakce měla atributy, které se mění spojitě podle diferenciálních rovnic vztázených k této transakci (v praxi: k třídě, jejíž je transakce instancí). Mohli bychom říci, že jazyky typu K_2 se liší od jazyků typu K_1 tím, že prvek nesoucí spojitě se měnící atributy nemusí být jeden a počet takovýchto prvků se může měnit – mohou vznikat a zanikat. Závislosti obou výše popsaných typů mohou ovšem existovat mezi transakcemi. Transakce bez spojitě se měnících atributů je vlastně degenerovaným případem transakcí se spojitě se měnícími atributy. Mezi tyto jazyky patřil první vůbec v historii navržený jazyk pro kombinovanou simulaci (z roku 1968 a asi nikdy neimplementovaný) a několik jazyků definovaných pomocí podtříd třídy SIMULATION v objektově orientovaném jazyku SIMULA.

Poznámka. Než si všimneme jazyků kombinovaného typu K_3 , musíme si uvědomit, že spojité vztahy mezi atributy (vyjádřené pomocí systémů diferenciálních rovnic) nastávají jen uvnitř aktivity D (v případě typu K_1), resp. uvnitř jednotlivých transakcí (v případě jazyků typu K_2). Vztahy mezi transakcemi navzájem a k ostatním prvkům se projevují vždy jen jako diskrétní události.

Toto pravidlo porušují jazyky typu K_3 : ty připouštějí, aby mezi sebou spojité reagovaly atributy různých prvků, což je převratné právě u transakcí.

Nechť a je atribut transakce P a b je atribut jiné transakce Q a oba jsou spojeny nějakým systémem diferenciálních rovnic (jež se ovšem obecně může vztahovat i k dalším atributům spojité se měnícím podle rovnic tohoto systému). Když transakce Q ze systému zmizí, zmizí i atribut b a uvedený systém rovnic přestane mít smysl. Musí být nahrazen jiným systémem. Něco podobného by mělo nastat i v tehdy, když transakce Q do simulovaného systému vstoupila a „zaangažovala svůj atribut b “ do nějakého systému diferenciálních rovnic: ten ovšem musel před tím mít jiný tvar, nemohl obsahovat b .

U typu $K3$ se tedy setkáváme se spojitými změnami podle diferenciálních rovnic, jejichž systém se v čase mění. Prostředky pro popis něčeho podobného zatím v naší civilizaci neexistují, avšak systémy, v nichž diskrétní změny způsobují kvalitativní změny pravidel pro spojité změny, kolmo nás existují, a to velmi často – příkladem z průmyslu může být hlubinná pec v ocelárnách, v níž jsou zahřívány bramy, jejichž počet – a tudíž i vzájemné přenášení tepla – se v čase mění. Z toho důvodu jsou jazyky typu $K3$ velmi aktuální. Proměnná pravidla pro spojité změny se v nich úspěšně obcházejí tak, že uživatel jazyka popisuje transakce jako zapojení prvků ideálního analogového počítače (sčítaček, integrátorů, násobiček, invertorů, generátorů funkcí atd.), který mohou v čase měnit svou strukturu (může i libovolně narůstat, pulzovat co do své velikosti apod.). Tyto jazyky jsou vybaveny i možností definovat makra (vybraná zapojení) a dynamicky je zapojovat všude, kde je možno zapojovat původní, „elementární“ prvky idealizovaného analogového počítače. Prvním jazykem tohoto typu byl NEDIS ukrajinské provenience, který se v něčem nechal inspirovat třídami jazyka SIMULA (ale ne podtřídami a lokálními třídami). Po něm následovalo několik jazyků budovaných přímo na bázi jazyka SIMULA (německý SIMKOM/S, norský COMBINEDSIMULATION, portugalský CDS a český BOSCO).

Jazyky typu $K3$ se dnes začínají používat také v aplikaci tzv. přímkové metody pro simulaci spojitých systémů popsataných pomocí parciálních diferenciálních rovnic. Zhruba řečeno, přímková metoda spočívá v diskretizaci takového systému, při níž se derivace podle jiných hodnot než času approximují diferencemi, takže z parciální rovnice vznikne soustava rovnic obyčejných. Vzhledem k tomu, že odstraněné parciální derivace mohou měnit své hodnoty v čase, je nutno měnit i počet approximujících obyčejných rovnic, jež však spolu interagují spojité, takže možnosti, které nabízejí jazyky typu $K3$, nelze v tomto případě obcházet pomocí nástrojů, jež nabízejí jazyky typu $K1$ a $K2$.

Jazyky pro OOP

Objektově orientované programování má úzký vztah k simulačním jazykům a tím i k (počítačové) simulaci. Vzniklo dokonce právě na podnět ze strany těchto jazyků, i když se nyní aplikuje univerzálně, i mimo simulaci. V této kapitole se s ním podrobněji seznámíme, avšak předem upozorňujeme, že k nejdůležitějšímu jeho systému z hlediska vývoje

programování i z hlediska simulace budou vypracovány detailní učební texty v blízké budoucnosti.

Jak už jsme naznačili, simulační jazyky mají umožnit svým uživatelům popisovat simulační modely v podstatě stejnou formou, jak by pro své kolegy v profesi popisovali zkoumaný systém. Důsledek, který z toho plyne, je nutnost mnoha simulačních jazyků, orientovaných na nejrůznější oblasti vědy, techniky a řízení, ba je to nutnost vzniku stále nových simulačních jazyků. To je neúnosná situace, odborníci v simulaci se snažili ji řešit, až to úspěšně dokázali v roce 1967 autoři jazyka SIMULA. Část myšlenek, které byly v tomto jazyku realizovány, po mnoha létech převzal celý svět a nazval je objektově orientované programování. Pak ovšem popularita tohoto způsobu programování způsobila jednak to, že z reklamních důvodů kdejaký autor softwaru prohlašoval, že jeho produkty jsou objektově orientované (nějaký objekt lze najít prakticky ve všem), a že to, co se z idejí jazyka SIMULA pod objektově orientované programování zahrnulo, byl jen zlomek a ten se nejvíce vzdálil právě od toho, co vznik objektově orientovaného programování podnítilo, totiž od simulace. Je tedy nutno objektově orientované programování nejprve vymezit podle toho, na čem se dnes shoduje odborná veřejnost, a pak ukázat na jeho vztah k simulaci a k jazyku SIMULA.

Objektově
orientované
programování

Třída
Atribut
Metoda
Instance třídy
Objekt

Zpráva
Selektor
Adresát

Ve shodě s názory přijatými dnes už drtivou většinou odborníků v celém světě můžeme vymezit pojem **objektově orientovaného programování** (v dalším OOP) jako proces tvorby počítačových programů s pomocí reprezentace znalostí ve formě tříd, to jest ve formě abstraktních entit, pro které platí následující tvrzení.

1. Třída obsahuje pojmenované (identifikovatelné) hodnoty čili **atributy** a algoritmy čili **metody**.
2. Třída je „prototypem“ libovolného, předem neurčeného množství konkrétních entit, tak zvaných **instancí třídy**. Místo o instanci se často mluví o objektu. Když někdo řekne, že je něco **objekt**, chce tím obvykle vyjádřit, že je to instance nějaké – snad nedůležité či v daném kontextu neznámé – třídy. O instanci, která je vytvořena z nějaké třídy *C*, říkáme, že je to **instance třídy C** nebo že **patří do C**.
3. Každá instance třídy dejme tomu *C* nese své vlastní atributy zavedené podle jejich definice v třídě *C* a na požádání od (jiných) instancí téže třídy nebo jiných tříd (tedy od jiných objektů) může provést jakoukoliv metodu v třídě formulovanou, pokud provádění takové metody nebylo v definici třídy *C* zablokováno (viz bod 5). Když nějaký objekt požádá instanci *X* aby provedla metodu *F*, říkáme, že **poslal zprávu** instanci *X* se **selektorem F** a že *X* je **adresátem** této zprávy. Anglicky se zpráva nazývá **message** a poslání zprávy se nazývá **message transfer**.
4. U každé třídy lze stanovit „privátní“ čili **chráněné** atributy: tato vlastnost se přenáší na atributy každé instance této třídy tak, že s nimi může manipulovat ona sama instance, když dostane zprávu, aby provedla nějakou metodu.
5. Stejně jako jsou chráněny některé atributy, mohou být chráněny i některé metody; ty pak může aplikovat instance, k níž jsou vztaženy,

- jen jako součást provádění jiných – případně i nechráněných – metod. Jinými slovy, chráněné metody mohou vystupovat pouze jako selektory zpráv, které posílá instance sama sobě.
6. Každá třída může být obsahově obohacena čili **specializována** přidáním nových atributů a formulováním nových metod. Takto specializovaná třída se stává novou třídou, tzv. **podtřídou** výchozí třídy. Výchozí třídu nazýváme **prefixem** podtřídy.
 7. Při formulaci třídy není nutno nic vědět o jejích případných podtřídách, ani o jejich případném počtu.
 8. Některé metody v třídě lze prohlásit za **virtuální**. Virtuální metoda je ta, o níž v definici třídy stanovíme, že má smysl, ale nemusíme stanovit, jaký tento smysl konkrétně je. Ten lze formulovat až v podtřídách, přičemž každý objekt interpretuje smysl takové virtuální metody podle toho, do které podtřídy patří.

Specializace

Podtřída

Prefix

Virtuální metody



Jak už jsme výše naznačili, je OOP lidskou programovací činností. Pod pojmem OOP tedy zahrnujeme jak tvorbu tříd, tak jejich aplikaci, a ta může být zaměřena jak na tvorbu dalších tříd, tak na použití tříd k tvorbě konkrétních programů. Těmito programy mohou být i počítačové modely, mimo jiné i modely simulační. **Soubor vhodně sklovených tříd může být definicí (simulačního) jazyka**. Jak už jsme uvedli, od každé třídy lze v OOP vytvořit libovolný počet instancí. (Je-li nějaký programovací prostředek omezen tak, že od některých jeho tříd lze vytvořit jen pevný počet instancí, nejde o prostředek zaměřený na OOP a jeho použití není OOP.) Použití OOP tedy vede k zavádění jazyků základního typu T, neboť aktivity v modelech sestavených s pomocí OOP neexistují, vždy jde o transakce, eventuelně degenerované tak, jak bylo naznačeno v závěru oddílu 7.2.5.

Dnes jsou v oblibě prostředky pro OOP amerického původu, jako je SmallTalk, C++, JAVA nebo novější verze Pascalu. Tyto prostředky vyházejí z více či méně radikálního pojetí OOP, dle něhož jediná možnost dění v počítači spočívá ve vysílání zpráv a v reakcích adresátů na ně. Kvaziparalelní systémy (viz závěr kapitoly 15) se v těchto jazycích nepřipouštějí. Při tomto přístupu se tvorba simulačních prostředků omezuje na prostředky s elementárními transakcemi (viz oddíl 15.7). Jinými slovy, při použití prostředků pro OOP, které jsou dnes rozšířeny, se sestavují konfigurace tříd, pomocí nichž jsou realizovány různé jazyky, v nichž existují jednak pasivní transakce (transakce odpovídající prvkům, které „pozorujeme“ na simulovaném systému, ale bez vlastních životních pravidel), jednak elementární transakce, které nesou útržky děje v systému. Pasivní transakce jsou instance tříd, zatímco metody, které tyto transakce provádějí (na podněty odjinud), odpovídají elementárním transakcím.

Můžeme říci, že jazyky takto implementované jsou jazyky typu TT_E . A na rozdíl od nich můžeme též jazyky, o nichž jsme se zmínili v odstavci 15.6, prohlásit za jazyky typu T. Jazyky typu TT_E spolu s jazyky typu T jsou totiž jedinými dvěma podskupinami jazyků základního typu T.

Pojmy k zapamatování:



- Jazyky
 - pro spojitu simulaci
 - typu A
 - typu AT
 - typu T
 - typu TA
 - základního typu A
 - základního typu AT
 - základního typu T
- Kvaziparalelní
 - programování
 - systém
- Prvky
 - aktivní
 - elementární
 - pasivní
- Systémy
 - typu A
 - typu AT
 - typu T



Kontrolní otázky:

1. Jaký je rozdíl mezi jazyky A, AT, T a TA?
2. Vysvětlete kvaziparalelní programování.



Shrnutí obsahu kapitoly

Simulační programovací jazyky (resp. – stručně – simulační jazyky) jsou důležité programovací nástroje pro vytváření simulačních modelů. Dělíme je podle toho, zda mohou zobrazovat systémy s aktivitami, s transakcemi, nebo s aktivitami i s transakcemi, na jazyky základního typu A (lze v nich popisovat jen systémy složené z aktivit), jazyky základního typu T (lze v nich popisovat jen systémy složené z transakcí) a jazyky základního typu AT (lze v nich popisovat systémy složené z transakcí a aktivit, většinou tak, že každá transakce během své existence v systému nějak interaguje – nebo alespoň požaduje interakci – s nějakou aktivitou. Jazyky typu základního AT pak dělíme ještě na jazyky typu AT a jazyky typu TA, a to podle toho, zda je popis dynamiky (změn stavů) systému vázán na transakce nebo na aktivity. Důležitou skupinou jazyků typu A jsou jazyky pro spojitu simulaci.

13 Organizace simulační studie, metody jejího vyhodnocení

V této kapitole se dozvítě:

- Řízení simulační studie.

Po jejím prostudování byste měli být schopni:

- navrhnout vlastní studii zvoleného objektu,
- umět vyhodnotit získaná data.

Klíčová slova této kapitoly:

Simulační studie.

Doba potřebná ke studiu: 4 hodiny

Průvodce studiem

V kapitole týkající se základních pojmů a v kapitole týkající se programovacích prostředků jsme se zabývali způsoby popisu simulovaného systému, který je v počítačovém systému přeložen a prováděn jako simulační pokus. Taktéž jsme zavedli i termín simulační studie pro cílenou posloupnost simulačních pokusů. Je evidentní, že popisu simulovaného systému může – a to v opakovaném režimu – odpovídat i simulační studie, avšak vzniká otázka, zda a jaké jsou prostředky, které usnadňují její programování, tj. které jsou zaměřeny na to, že simulační studie je cílená, že neopakuje simulační pokusy neusporeádaně, nýbrž kvůli jistému záměru. Je evidentní, že – podobně jako simulované systémy – mohou i cíle simulační studie pokrývat široké spektrum možností. V rámci přípravy simulační studie využije všechny nabité znalosti v rámci tohoto předmětu.



13.1 Řízení simulační studie

Než postoupíme ve výkladu dále, uvedeme příklad, který pak zobecníme. Předpokládejme, že chceme najít nějakou optimální strukturu továrny, která má být postavena. Simulační studie tedy bude složena ze simulačních pokusů, z nichž každý bude odpovídat jedné variantě továrny. Není ani vyloučeno, že simulační studie bude organizována tak, že se z dosavadních simulačních pokusů bude „učit“ a bude postupně navrhovat další varianty továrny, které by mely být stále lepší a lepší (což je ovšem vždy otázka, na kterou dá odpověď právě simulační pokus s danou variantou). Když popisujeme simulační pokus, popisujeme v simulačním jazyku továrnu. Kdybychom však chtěli podobným způsobem popsat simulační studii, museli bychom popisovat situaci, v níž jako by byla realizována jedna továrna (tj. první varianta, odpovídající prvnímu simulačnímu pokusu), s ní by se mely udělat nějaké zkušenosti a pak by se měla likvidovat, na základě zmíněných zkušeností by se měla realizovat jiná, snad lepší továrna (druhá varianta, odpovídající druhému simulačnímu pokusu), po získání zkušeností s ní by se měla na jejím místě realizovat další továrna a tak dále, dokud se nepostaví továrna vskutku perfektní. Podobně by se měla popsat např. si-



mulační studie týkající se nádorové terapie (pacient je léčen jedním způsobem, pak zlikvidován, znova oživen ve stavu před léčbou a na základě zkušeností s první léčbou léčen jinak atd.).

Ačkoliv na počítači lze nasimulovat ledacos, i výše uvedené drastické procesy, je evidentní, že uživatele simulačních programovacích prostředků, kteří nejsou odborníky v informatici a mají představy, které se shodují s realitou jejich profese, by nutnost představ takových drastických procesů spíše odradila a znechutila, než aby jim pomohla. Celá záležitost je navíc komplikovaná ještě tím, že simulační pokus začíná ve všech simulačních prostředcích časem rovným nule a na tomto předpokladu jsou založeny takřka všechny pomocné výpočty zabudované do simulačních jazyků (hodnocení a aktualizace histogramů, výpočty průměrů hodnot, které se mění v čase atd.). Takže při likvidaci jedné varianty továrny, pacienta či jiného simulovaného objektu a jeho následného vytvoření s jinými vlastnostmi by bylo třeba, aby uživatel zlikvidoval a znova vytvořil zmíněné histogramy a další pomocné entity a také posunul časovou osu tak, aby další varianta simulovaného systému začala existovat v čase nula a nikoliv v čase, kdy byla likvidována varianta předcházející.

Je tedy zřejmé, že právě popsaný přístup by uživateli mnoho nepomohl, a tak je používán jen zřídka, např. v simulačních prostředcích vybudovaných na bázi populárních jazyků pro OOP.



Shrňme obecné poznatky z předcházejícího příkladu. Simulovaný systém popisujeme jako něco, o čem předpokládáme, že by to mohlo reálně existovat, dokonce v newtonovském čase. Když chceme takový popis rozšířit na simulační studii, dostáváme se kam s mimo fyzikální svět, v němž simulovaný systém může existovat; dostáváme se do světa, kde neplyne čas tak, jak jsme zvyklí a jak to aplikujeme i ve všech oborech vědy, techniky i řízení společnosti s výjimkou několika oblastí moderní fyziky. V takovém světě už např. druhý simulační pokus studie předpokládá, že popsaný systém z prvního pokusu nenávratně zmizí, čas se vrátí, vznikne jiný systém – možná ovšem s vlastnostmi, které získal na základě „poučení“ z dějů, které nastaly v prvním systému – a ten se začne nějak v čase měnit. Návrat času (resp. zcela nový čas) i poučení z „minulé budoucnosti“ (např. ze stavu systému z prvního pokusu, který nastal v okamžiku, do kterého druhý pokus dosud nedospěl) drasticky nesouhlasí s vlastnostmi fyzického světa, které jsou zahrnuty do sémantiky simulačního jazyka a které obsahují vztahy, na nichž jsou založeny tak běžné faktory myšlení jako kauzalita a běh newtonovsky chápání času. Aby dostali uživatelé simulačních prostředků možnost popsat řízení simulační studie tak „hmatatelně“, jako mohou popisovat simulační pokus, museli by své „vidění světa“ rozšířit kam s mimo ten fyzikální svět, v němž simulovaný systém může existovat, museli by být vedeni např. k představám, že je nějaká metafyzicky vysoko postavená bytost, která si pohrává s různými fyzickými světy, z nichž některé mohou případně i existovat paralelně vedle sebe, ruší tyto světy (včetně jejich času a objektů, které jsou v nich) a nahrazuje jinými a přitom je porovnává, dokud nevybere ten nejlepší.

Přestože něco podobného cítil ve svých filozofických představách již v 17. století jeden ze zakladatelů infinitesimálního počtu G. W. Leibniz, tvůrci simulačních jazyků se takovýmto představ obávali, a tak pro řízení simulační studie nabízeli více nebo méně jednoduché algoritmické prostředky. Z nich nejjednodušší vedou uživatele k tomu, že sestaví jakousi tabulku parametrů a simulační studie postupuje tak, že provede první pokus s parametry z první řádky tabulky, pak druhý pokus s parametry z druhé řádky tabulky a tak pokračuje, dokud tabulku nevyčerpá. Výsledky jednotlivých pokusů řadí do histogramů a počítá z nich průměry a maxima. Nejsložitější prostředky vedou naopak uživatele k tomu, že chápe simulační studii jako algoritmizovaný cyklus, v němž vystupuje simulační pokus jako jedna složka, a to dle následujícího schématu:

1. příprava parametrů prvního simulačního pokusu;



- začátek cyklu: 2. provedení simulačního pokusu;
3. zhodnocení výsledků simulačního pokusu;
4. rozhodnutí, zda ve studii pokračovat; pokud ano:
5. příprava parametrů pro další simulační pokus;
6. skok na začátek cyklu.

Kroky 3 až 5 mohou být velmi složité a mohou používat „globální“ proměnné, tj. proměnné, které nejsou svázány s žádným jednotlivým simulačním pokusem. Na ně se mimo jiné mohou ukládat výsledky provedených simulačních pokusů i parametry, které další pokusy ovlivní a pomocí těchto proměnných lze jednotlivé simulační pokusy navzájem porovnávat.

Avšak představu oné bytosti, jež by byla nadřazena různým světům, které navzájem fyzicky nesouvisejí, ale jsou v mysli této bytosti porovnávány, tvořeny a likvidovány, je přece jen možno realizovat, a to např. v objektově orientovaném jazyku SIMULA. V něm totiž existují dva druhy hierarchií, které v jiných simulačních ani objektově orientovaných jazycích nejsou.

- Hierarchie do sebe vnořených tříd: třída může kromě atributů, metod a životních pravidel obsahovat také definice jiných tříd. Taková třída se nazývá **hlavní třídou** a její instance představují **inteligentní činitele** (angl. **intelligent agents**), jež „myslí“ a přitom používají pojmu zobrazených třídami obsaženými uvnitř této hlavní třídy (je nutno upozornit, že do českého jazyka pronikl i termín **inteligentní agent**).
- Hierarchie kvaziparalelních systémů: jeden kvaziparalelní systém může obsahovat prvky, které samy obsahují kvaziparalelní systémy.

Hlavní třída

Inteligentní činitel
(Inteligentní agent)

Jelikož prostředky prvního druhu umožňují modelovat systémy složené z prvků, které „myslí“, umožňují modelovat i systémy složené z inteligentních činitelů, které si představují nějaké složité děje. Kombinace těchto prostředků s prostředky druhého druhu umožňuje modelovat systémy, jejichž inteligentní činitelé si představují děje, jejichž složitost vzniká paralelním vzájemným ovlivňováním v čase. Jinými slovy, takoví intelligentní činitelé jsou modelováni, jako by simulovali. V jazyku SIMULA lze tedy

simulovat (či jiným způsobem modelovat) systémy složené z prvků, z kterých některé či všechny mají své „soukromé“ simulační modely, vzájemně na sobě nezávislé, a manipulují s nimi.

S pomocí jazyka SIMULA lze tedy simulační studii chápát, popsat a na počítači realizovat jako dynamický systém, jehož prvky jsou jednotlivé simulační pokusy. Prostředky pro OOP lze aplikovat nejen na úroveň simulačních pokusů, ale i na úroveň simulační studie, jejíž složitost tedy může nabývat extrémní výše.

V anglosaské literatuře, která pojednává o obecných prostředcích pro simulaci, se můžeme často setkat s termínem **experimental frame**. (Český ekvivalent **experimentální rámec** se u nás příliš neujal.) Tento termín vyjadřuje obecně činnost vztaženou k automatizaci experimentování se simulačním modelem. Často pod tímto termínem myslí autoři právě řízení simulační studie, avšak naplatí to obecně. Slova *experimental frame* jsou totiž libovolně interpretována v plném významu, co znamenají v angličtině, takže někdy je autoři chápou i např. jako formulaci výstupu výsledků během simulačních pokusů. V osmdesátých letech totiž vznikly tendenze oddělit popis vlastního simulovaného systému od popisu experimentování s jeho simulačním modelem a pro každý z popisů použít jiného programovacího jazyka. Popis simulovaného systému sice odpovídá – jak jsme zdůraznili v kapitole 15 – popisu simulačního pokusu, ale během takových pokusů se může vyskytnout výstup výsledků, aktualizace histogramů, sběr a integrování některých parametrů v simulovaném čase apod. a popis těchto akcí měl být oddělen od popisu vlastního systému a připojen k popisu řízení simulační studie.



Uveděme příklad. Při simulaci továrny popisujeme simulační pokus tak, že popíšeme onu továrnu včetně toho, jako bychom během její existence přímo v ní dělali nějaké pokusy, měření, detekce atd. Byly tendenze oddělit vlastní popis továrny od popisu těch pokusů, měření a detekcí prováděných v ní a tyto akce připojit k popisu simulační studie, tj. k popisu strategie, jak např. na základě údajů o chování jedné varianty továrny vytvořit variantu jinou, snad lepší. Z jistého pohledu na věc byla idea oddělení popisu experimentů od popisu továrny vhodná: stává se totiž, že na základě nějakých výsledků pokusů s továrnou nás napadne to, že bychom s ní ještě měli udělat jiné pokusy, než ji změníme, tj. než přejdeme na jinou variantu. Kdyby byl popis experimentování oddělen od popisu továrny, změnili bychom v takovém případě pouze ten popis experimentování.

Avšak chtít se vyjadřovat podobným jazykem jak o experimentování se systémem během jeho existence, tak o vytváření dalších variant systému je velmi násilný přístup. Nadto virtuální metody vyřešily problém výměny experimentálních kroků v popisu simulovaného systému. Možná místa experimentů v systému se prohlásí za virtuální procedury a jejich náplň se pak může libovolně měnit, a to odděleně od popisu simulovaného systému (a bez zásahů do něho). A tak experimentální rámec jako cosi svébytného a s vlastním jazykem se udržel jen u některých simulačních jazyků, z nichž nejznámější v USA je DYMOLA.

Pojmy k zapamatování:



- Atribut
- Instance třídy
- Metoda
 - virtuální
- Objekt
- Objektově orientované programování
- Podtřída
- Prefix
- Specializace
- Třída
 - hlavní

Shrnutí:

Objektově orientované programování je moderní způsob tvorby programového vybavení. Hodí se nejen pro vytváření programů, ale i pro implementaci nových programovacích jazyků, aniž by byly vyžadovány jejich speciální komplátory. Potíž je v tom, že skoro žádný prostředek pro objektově orientované programování nemá možnost zavedení kvaziparalelního systému, a tak implementace jazyků typu AT, TA a T je v těchto prostředcích prakticky nemožná. SIMULA je v tomto ohledu výjimkou a ukazuje se jako výborný objektově orientovaný prostředek i pro definice nových simulačních jazyků (tj. jazyků s novými univerzemi sémantiky, např. pro nové aplikační obory) i pro vytváření simulačních modelů.



Literatura



(Dahl a Nygaard 1967) Dahl, O.-J., Nygaard, K.: *SIMULA, a language for programming and description of discrete event systems*. 5. vydání. Oslo: Norsk Regnesentralen, 1967.

(Dorda a Teichmann 2012) Dorda, M.; Teichmann, D.: About a Modification of E_r/E_s/1/m Queuing System Subject to Breakdowns. In *Proceedings of 30th International Conference Mathematical Methods in Economics 2012*, Karviná, 11. – 13. září 2012, s. 117-122. ISBN 978-80-7248-779-0.

(FELD 2013) Kendallová klasifikace obsluhových systémů. *FELD Access server* [online]. [cit. 2013-10-18]. Dostupné z: <http://access.feld.cvut.cz/view.php?cisloclanku=2005111601>

(Gershenson 2002) Gershenson, C.: Complex philosophy. In Proceedings of the 1st Biennial Seminar on Philosophical, Methodological & Epistemological Implications of Complexity Theory. La Habana, Cuba. [cit. 2012-02-12], URL <http://uk.arxiv.org/abs/nlin/0109001v3>, 2002

(Hájek a Havránek 1978) Hájek, P., Havránek, T. *Mechanizing Hypothesis Formation – Mathematical Foundations of a General Theory*. Berlin: Springer Verlag, 1978.

(Kendall 1953) Kendall, D. G.: Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded Markov chain. In *The Annals of Mathematical Statistics* Vol.24, 1953, s. 338–354.

(Kindler 1969) Kindler, E.: Simulation System COSMO – Description of Its Language and Compiler. *Kybernetika*, 1969, roč. 5, s. 287.

(Kindler 1980) Kindler, E. *Simulační programovací jazyky*. Praha: SNTL, 1980. 280 s.

(Kindler 2004) Kindler, E.: Algoritmizace v jazyku Simula. distanční opora, PřF, OU, Ostrava, 2004.

(Kolektiv autorů 1986) Kolektiv autorů. Dohoda o chápání pojmu simulace systémů. *Automatizace*, 1986, roč. 29, č. 12, s. 299–300.

(Kotva 1979) Kotva, M.: *Obecná metoda multikompartimentových modelů*. Kandidátská disertace. Praha: VŠE, 1979.

(Křivý 1986) Křivý, I.: *Obecná metoda multikompartimentových modelů*. Výzkumná zpráva k úkolu ZV I-1-5/04. Ostrava: Pedagogická fakulta, 1986.

(Křivý a Kindler 2003a) Křivý, I., Kindler, E.: Simulace a modelování 1. distanční opora, PřF, OU, Ostrava, 2003.

(Křivý a Kindler 2003b) Křivý, I., Kindler, E.: Simulace a modelování 2. distanční opora, PřF, OU, Ostrava, 2003.

- (Malík 1989) Malík, M. *Počítačová simulace*. Skripta MFF UK. Praha: UK Praka, 1989. 535 s. ISBN 80-7066-121-6.
- (Marek a Schreiber 1984) Marek, M., Schreiber.: I. *Stochastické chování deterministických systémů*. Praha: Academia, 1984. 160 s.
- (Mladov 1966) Mladov, A. G.: *Sistemy differencialnych uravnenij i ustojčivost po Ljapunovu*. Moskva: Vysšaja škola, 1966. 272 s.
- (Nahodil 2007) Nahodil, P.: Od jednoduchých robotů k anticipujícím stvořením. In: Mařík V., Štěpánková O., Lažanský J. a kol.: Umělá inteligence 5. Academia, Praha 2007. ISBN 978-80-200-1470-2.
- (Nekvinda et al. 1976) Nekvinda, M., Šrubař, J., Vild, J.: *Úvod do numerické matematiky*. Praha: SNTL, 1976. 288 s.
- (Pelánek 2011) Pelánek, R.: Modelování a simulace komplexních systémů, Masaryk University, 236 p., 2011. ISBN: 978-80-210-5318-2.
- (Rábová et al. 1992) Rábová, Z., et al.: *Modelování a simulace*. Skripta FEL VUT Brno. Brno: VUT Brno, 1992.
- (Ralston 1978) Ralston, A.: *Základy numerické matematiky*. Praha: Academia, 1978. 636 s.
- (Rosen 1970) Rosen, R.: *Dynamical System Theory in Biology*. Vol. I. *Stability Theory and Its Applications*. New York: Wiley – Interscience, 1970.
- (Schriber 1991) Schriber, T. J. *An Introduction to Simulation Using GPSS/H*. New York: Wiley & Sons, 1991.
- (Simula 1989) SIMULA Standard as defined by the SIMULA Standard Group, 25th August 1986. Oslo: Simula a. s., 1989.
- (Štěrba) Štěrba Ladislav: Teorie front, Xstel37, elektronický dokument. Datum neznámý.