M. Malík; Zdeněk Benda
Počítačová simulace [Dokončení]

Pokroky matematiky, fyziky a astronomie, Vol. 29 (1984), No. 2, 61--80

Persistent URL: http://dml.cz/dmlcz/139979

Počítačová simulace*)

Marek Malík, Praha, Zdeněk Benda, Brno

4. PROCESOVÁ REALIZACE SIMULAČNÍHO MODELU

Vlastní programová realizace simulačních procesů je poměrně náročná. Programovací jazyk, ve kterém má být realizován simulační program na procesovém principu, by měl mít alespoň základní prostředky pseudoparalelního či kvaziparalelního programování (tj. konstruce umožňující pozastavení a opětovné spuštění samostatných programových jednotek). Pomocí takových prostředků pak není realizace simulačních procesů příliš složitá. Každý exemplář procesu se realizuje jako samostatný koprogram a potlačovací prostředky jako speciální struktury řízení programu (např. procedury), které jednak využívají prostředky paralelismu a jednak zasahují do údajové struktury řídícího simulačního kalendáře, která registruje plány procesů.

4.1. Příklad procesové realizace

Ilustrujme nyní problematiku simulačních procesů na příkladě „Písek" a ukažme si, jak je možno celý děj systému modelovat pomocí simulačních procesů.

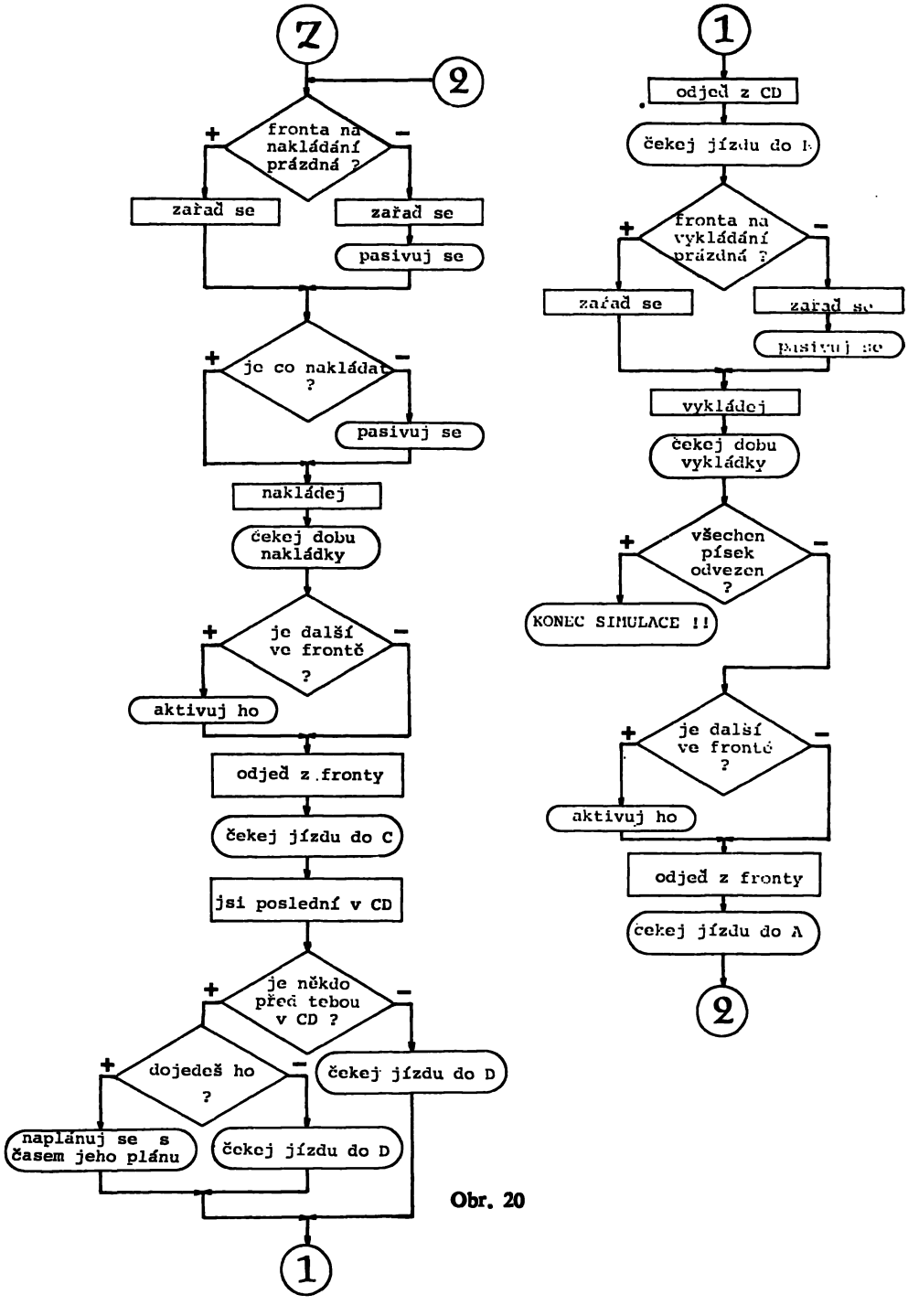
Jednou z možností je shrnout celý děj týkající se jednoho auta do jednoho procesu. Dostaneme tak jeden prototyp procesu pro všechna auta v modelovaném systému a pro každé konkrétní auto vytvoříme v programu jeden exemplář procesu podle tohoto prototypu.

Schematický vývojový diagram procesu auta je na obrázku 20. (Oválnými značkami označujeme příkazy působící potlačení procesu.)

Výpočet simulačního modelu zahájíme vytvořením všech exemplářů procesů a všechny převedeme postupně do aktivního stavu. Podrobněji: vytvoříme exempláře procesů a všechny naplánujeme k aktivaci a časem 0 (počáteční bod uvažované časové osy systému). V kalendáři jsou tedy nejprve plány všech procesů aut se stejným časem aktivace. V tomto okamžiku začne pracovat řídící struktura (jak je to možno realizovat, ukážeme si později), která aktivuje proces s plánem na začátku kalendáře (nechť je to proces p_1). Proces p_1 spustí svůj výpočet, zařadí se do fronty na nakládání, a protože fronta byla před tím prázdná, pokračuje ve výpočtu, až se suspenduje na potřebnou dobu nakládky. Tím se jeho plán přesune za plány všech ostatních procesů (ty mají čas nula) a řídící struktura spustí další proces, který má plán nyní „na začátku" kalendáře (označme si jej p_2). Tento proces stejně jako před tím p_1 spustí svůj výpočet, zařadí se do fronty na nakládání, ale protože ta již nebyla prázdná — je v ní proces p_1 — přejde do pasívního stavu — tj. jeho plán se vyřadí z kalendáře. Řídící struktura pak spustí další proces, který se podobně jako p_2 zařadí do fronty na nakládání a přejde do pasívního stavu; totéž se bude dít se všemi dalšími procesy aut původně naplánovanými v čase 0.

V kalendáři pak zůstane pouze plán procesu p_1, ale nikoli již s časem nula, ale s časem konce nakládky prvního auta. Všechny ostatní procesy aut jsou v pasívním stavu (neplánovány) a jsou zařazeny v datové struktuře (např. spojovém seznamu), která realizuje frontu aut na nakládání.

*) Dokončení článku, jehož první část byla otištěna v minulém čísle na str. 1—29.



Obr. 20

Řídící struktura nyní zvýší hodnotu simulárního času na čas plánu procesu p_1 a spustí tento proces. (V podstatě jsme v simulárním čase „přeskočili" dobu, po kterou se nakládalo první auto a celý

systém se — z našeho pohledu — neměnil.) Proces p_1 pokračuje ve svém výpočtu — aktivuje k okamžitému provedení svého následníka ve frontě (to je proces p_2). V kalendáři jsou po této aktivaci dva plány procesů p_2 a p_1 (v tomto pořadí), a to se stejným časem plánu*).

Řídící struktura tedy spustí proces p_2, který pokračuje ve výpočtu od místa svého posledního potlačení — začne nakládat (první auto patrně celou hromadu nenaložilo) a suspenduje se po dobu nakládky odpovídajícího auta (jeho plán se v kalendáři posune dále o tuto dobu). Tím je opět p_1 prvním procesem „na řadě", řídící struktura jej spustí; proces p_1 opustí frontu na nakládání a suspenduje se po dobu jízdy prvního auta z místa „**A**" do místa „**C**".

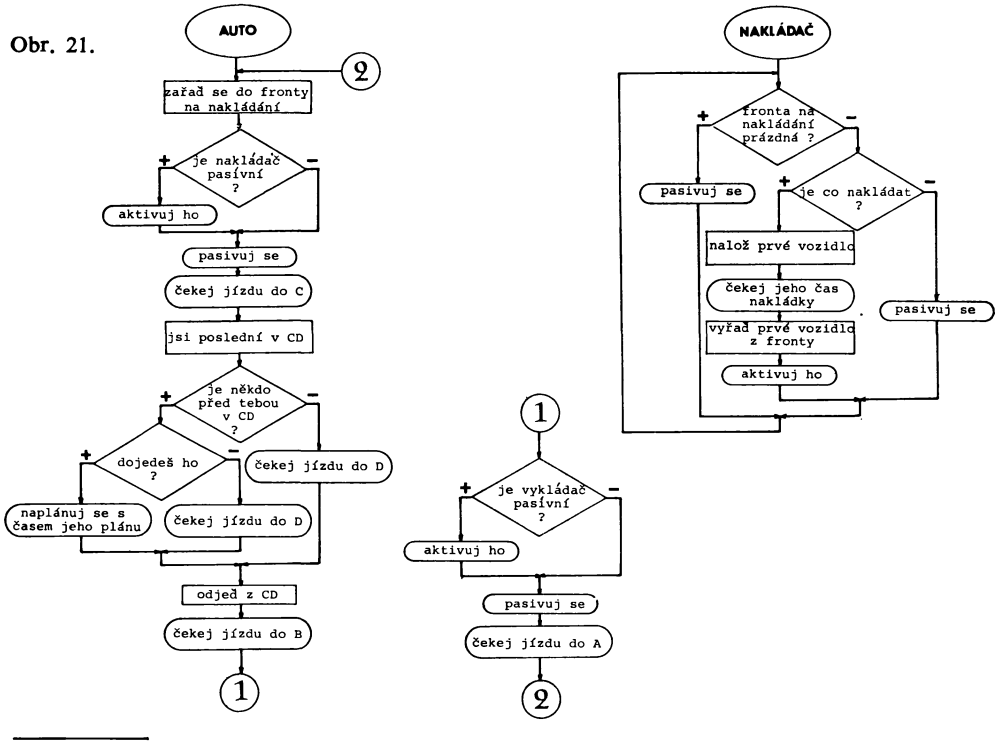
Simulační kalendář pak vybere minimum z časů plánů procesů p_2 a p_1 (ostatní procesy jsou stále pasivní ve frontě na nakládání), na zjištěnou dobu zvýší simulární čas (opět „přeskočíme" dobu, po kterou se systém vlastně neměnil) a spustí příslušný proces.

Analogickým způsobem pak pokračuje celý výpočet dále.

Navržená a právě diskutovaná realizace simulačního modelu pomocí procesů jediného typu není ovšem jediná možná. V místě „**A**" lze např. uvažovat zvláštní nakládač provádějící nakládku vozidel a podobně v místě „**B**" vykládač, který písek z aut vykládá, a ty lze realizovat v modelu pomocí zvláštních procesů. (Takovou úvahu můžeme provést celkem bez ohledu na to, zda ve skutečném systému zmíněná technologická zařízení skutečně existují či ne.)

Provedená dekompozice systému na elementárnější celky přispívá k tomu, že jednotlivé procesy jsou (co se jejich vnitřní struktury týče) jednodušší a vzniklý program je přehlednější.

Obr. 21.



*) Stále je to čas konce nakládky prvního auta.

Na obrázcích 21 a 22 jsou uvedeny vývojové diagramy procesů auta a nakládače v právě uvedeném pojetí modelu (proces vykládače je velmi podobný nakládači).

5. SIMULAČNÍ SOFTWARE

5.1. Třídění modelů

Struktura simulačního programu je velmi výrazně poznamenána naším pohledem na modelovaný originál — především tím, zda modelovaný systém klasifikujeme jako spojitý nebo diskrétní nebo kombinovaného typu.

Zmíněná klasifikace záleží v tom, zda časovou množinu modelovaného systému považujeme při jeho analýze za spojité kontinuum nebo za konečnou množinu či za množinu jiných vlastností (tedy většinou kombinaci obou právě jmenovaných typů — odtud název kombinovaný systém). Zde je však nutno zdůraznit slovo „považujeme", neboť rozhodující pro tvar simulačního programu není ani tak modelovaný originál sám, jako jeho analyzovaný popis (připomeňme si obrázek 4) a ten je v řadě případů možné vytvořit různými způsoby podle toho, pro jakou úroveň abstrakce jsme se rozhodli nebo které prostředky popisu pokládáme za nejvhodnější.

Uvažujme třebas v našem příkladě „Fotosyntéza" transportní strukturu složenou z většího počtu jednotlivých řetízků identické vnitřní stavby podle obr. 23, kde **C**, **P**, **A** jsou jednotlivé biochemické typy elementů a k_i přechodové konstanty udávající např. negativně exponenciální rozdělení čekacích dob. (Neuvažované kraje řetízků považujeme za stále obsazené; resp. stále volné.)

Tento systém můžeme považovat (jak jsme ostatně činili v podobných případech dosud) za diskrétní — jeho časovou množinu budeme chápat jako souhrn všech okamžiků, kdy dojde k přeskoku elektronu na nějaké přechodové cestě některého řetízku. Simulační model pak vybudujeme metodou plánování událostí či plánování jednoduchých procesů iterujících událost přeskoku. Na základě výpočtu modelu pak můžeme stanovit výsledky simulace — totiž tvar funkcí

$$\mathcal{O}_{\mathbf{C}}(t) = \mathscr{E}_{\mathbf{C}}^{-}(t)/\mathscr{E}_{\mathbf{C}};\ \mathcal{O}_{\mathbf{P}}(t) = \mathscr{E}_{\mathbf{P}}^{-}(t)/\mathscr{E}_{\mathbf{P}};\ \mathcal{O}_{\mathbf{A}}(t) = \mathscr{E}_{\mathbf{A}}^{-}(t)/\mathscr{E}_{\mathbf{A}},$$

kde $\mathscr{E}_{\mathbf{S}}^{-}(t)$ je počet elementů typu „**S**" obsazených v čase t elektronem a $\mathscr{E}_{\mathbf{S}}^{-}$ je počet všech elementů typu „**S**".

Na tentýž systém se můžeme dívat spojitě — uvažovat jen nepřetržitý tok elektronů celou transportní strukturou a popsat jej soustavou diferenciálních rovnic:

$$d\mathcal{O}_{\mathbf{C}}/dt = k_0(1 - \mathcal{O}_{\mathbf{C}}) - k_1\mathcal{O}_{\mathbf{C}}(1 - \mathcal{O}_{\mathbf{P}})$$

$$d\mathcal{O}_{\mathbf{P}}/dt = k_1\mathcal{O}_{\mathbf{C}}(1 - \mathcal{O}_{\mathbf{P}}) - k_2\mathcal{O}_{\mathbf{P}}(1 - \mathcal{O}_{\mathbf{A}})$$

$$d\mathcal{O}_{\mathbf{A}}/dt = k_2\mathcal{O}_{\mathbf{P}}(1 - \mathcal{O}_{\mathbf{A}}) - k_3\mathcal{O}_{\mathbf{A}}$$

— viz [17], a tuto soustavu řešit nějakou vhodnou numerickou metodou.

Výsledky z obou přístupů jsou (zanedbáme-li rozdíly působené numerickými nepřesnostmi výpočtu) samozřejmě stejné.

V uvedeném případě je celkem evidentní, že vypracování řešení pomocí spojitého

popisu systému je výrazně méně pracné. Postačí však, abychom dostatečným způsobem pozměnili uvažovanou transportní strukturu a bude tomu právě naopak.

Zavedená klasifikace analytických přístupů (resp. terminologie) se přenáší i do simulačních modelů, takže mluvíme o modelech spojitých, diskrétních a kombinovaných přesto, že časová množina modelujícího systému — programu — je vždy konečná.

Obr. 23.

5.1.1. Modely spojitých systémů

Popis typických spojitých systémů je formulován pomocí diferenciálních rovnic. Kontinuum tvořící časovou množinu modelovaného systému je zobrazeno v diskrétním čase simulačního programu, který je kvantován tak „hustě", aby numerické řešení diferenciální rovnice prováděné programem vyhovovalo požadavkům na přesnost a stabilitu (pokud toho lze vůbec dosáhnout). Stavové změny, v originálu popsané diferenciální rovnicí, se v programu realizují nějakým operátorem numerické integrace použitým pro řešení dané rovnice. Kalendář událostí mívá v těchto programech triviální formu: uchovává pouze okamžitý čas, který je zvyšován o délku kroku.

Hlavní problémy, se kterými se při tvorbě spojitých modelů setkáváme, byly již v podstatě naznačeny:

— volba numerické integrační metody,

— řízení délky časového kroku se zřetelem na přesnost a stabilitu řešení při simulačním běhu (tj. velikost časového kvantování),

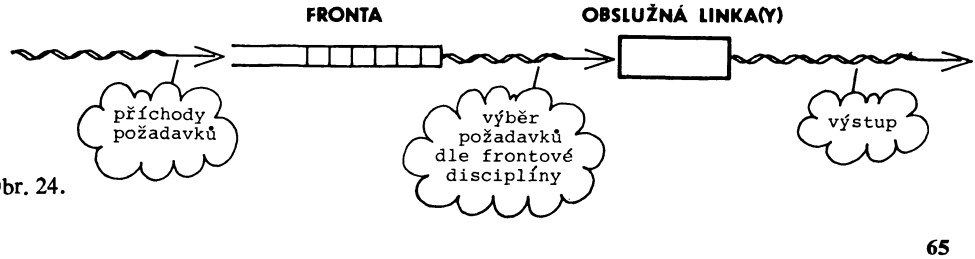
— spotřeba strojového času, která často limituje plnění předchozího požadavku.

Vedle zmíněného případu naší „Fotosyntézy" uveďme jako příklady spojitých simulačních úloh (z mnoha různých oborů, které zde nelze všechny vyjmenovat) problémy z pružnosti a pevnosti (deformace a kmitání složitých konstrukcí), řešení nelineárních elektrických obvodů, problémy z aerodynamiky a hydrodynamiky a další.

Z našich příkladů lze pro ilustraci použít ještě „Křižovatku", kde pohyb rozjíždějícího se vozidla je typickým (i když poměrně jednoduchým) problémem spojité simulace.

5.1.2. Modely diskrétních systémů

Nejčastějším případem použití diskrétních modelů jsou aplikace systémů hromadné obsluhy (obr. 24) a obslužných sítí.



Obr. 24.

Systémy hromadné obsluhy jsou dále klasifikovány podle typů rozložení příchodů požadavků, rozložení doby obsluhy a počtu obslužných linek (Kendallova klasifikace — viz. [22]). Pro některé typy problémů je známo analytické řešení (byť velmi složité); valná většina praktických problémů, jejichž řešení se převádí na simulaci těchto systémů, vede však ke strukturám takové složitosti, že se již analytickému řešení vymykají. Jsou to systémy s komplikovanějšími frontovými disciplínami, ztrátami požadavků, přerušováním obsluhy, prioritními požadavky a podobně. Simulační modely zde pak poskytují velmi účinné nástroje řešení.

Pro simulační modely těchto systémů jsou typické tyto rysy:

— proměnný počet prvků systému díky přicházejícím a odcházejícím požadavkům;

— reprezentace front pomocí spojových seznamů;

— vysoký stupeň paralelnosti — každý požadavek může být popsán procesem; běžné jsou i systémy, ve kterých se současně vyskytuje řádově $10^3 - 10^4$ požadavků;

— z paralelnosti plynoucí velké nároky na řízení programu simulačním kalendářem; vzhledem k tomu, že vlastní stavové změny jsou obvykle poměrně jednoduché, připadá značná část strojového času na práci zpřístupňujících procedur kalendáře;

— při velkém počtu objektů vyskytujících se v modelovaném systému zpravidla vznikají velké nároky na paměťový prostor.

Z našich příkladů je dosti typickým představitelem této skupiny úloha „Písek" (její systém je však — co se průchodu „požadavků" systémem týče — uzavřen).

5.1.3. Modely kombinovaných systémů

Posledním případem zavedené klasifikace jsou modely kombinované, ve kterých se vyskytují současně problémy obou výše uvedených skupin a do jisté míry též problémy vyplývající z nutnosti synchronizovat spojité a diskrétní části modelu.

Na systémy realizující kombinované simulační modely jsou přesněji řečeno kladeny mimo jiné tyto požadavky:

1. Podobně jako v případě spojitých modelů je třeba synchronizovat (vzhledem k možným vzájemným vztahům) časové kroky jednotlivých procesů spojité části systému, to jest procesů iterujících „událost" jednoho kroku příslušného numerického výpočtu.

2. Musíme též synchronizovat časový krok spojitých procesů (opět s ohledem na eventuální vzájemné ovlivnění) s plány diskrétních procesů plánovaných na přesný časový okamžik. To znamená, že potřebujeme, aby se diskrétní i spojité procesy „sešly" v jednom okamžiku simulárního času — diskrétně plánovaný proces zde např. může zasáhnout do parametrů diferenciální rovnice, kterou řeší algoritmus spojitého procesu.

Celá problematika je podrobněji popsána např. v [5, 6].

Kombinovaným modelem může být např. řešen náš příklad „Křižovatka" se spojitě pracujícími částmi pro modelování jízdy aut a diskrétně pracující částí modelující signalizační zařízení.

5.2. Přehled některých simulačních programovacích jazyků

Programovacích jazyků a programovacích systémů určených pro simulaci nebo k simulaci často používaných je nesmírná řada; jsou mezi nimi jak jazyky úzce speciální (např. čistě pro modelování telefonních ústředen), tak poměrně univerzální pro širší okruh problémů. Byly též učiněny (s různým zdarem) pokusy zavést do této spousty jazyků řád a klasifikaci (uveďme alespoň [13]). Domníváme se však, že rozebírat různé přístupy, které vedly k vytvoření různých simulačních jazyků, není ani zdaleka účelem tohoto článku.

Chceme zde proto ukázat jen základní rysy několika programovacích jazyků, s nimiž se lze setkat v simulačním programování.

5.2.1. Jazyky spojité

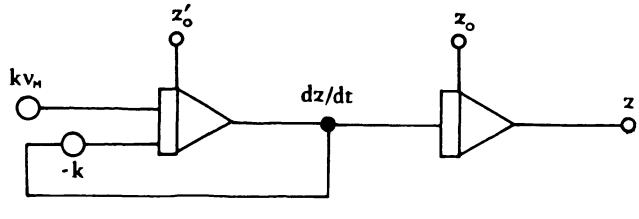
Z velké skupiny jazyků pro spojitou simulaci jsme vybrali jazyk *CSMP/360* (*C*ontinuos *S*ystem *M*odeling *P*rogram) a ukážeme jen zhruba princip jeho použití na řešení části příkladu „Křižovatka".

Jazyk *CSMP/360* využívá popisu modelovaného systému ve formě standardních funkčních bloků, kterými jsou např. integrátor, generátor funkce, omezovač, porovnávací blok a další (používané stejným způsobem jako pro analogové počítače). Modelovaný systém se nejprve popíše formou blokového schématu, které se pak přepisuje podle jistých pravidel do textové formy programu. Po očíslování bloků se zadává jejich typ, parametry a propojení.

Například řešení diferenciální rovnice rozjezdu vozidla z příkladu „Křižovatka":

$$\mathrm{d}z_p^2/\mathrm{d}t^2 = k_p(v_M - \mathrm{d}z_p/\mathrm{d}t),$$

kde v_M je maximální rychlost vozidel, z_p je poloha vozidla **p** na některé z os křižovatky a k_p parametr jeho akcelerace, může být popsáno blokovým schématem na obrázku 25, kde jsou použity pouze dva typy bloků, a to integrátor s počáteční podmínkou a konstanta.



Obr. 25.

5.2.2. Jazyky diskrétní

Z rovněž velmi rozsáhlé skupiny jazyků používaných pro diskrétní simulaci jsme vybrali tři: *GPSS*, *SIMSCRIPT* a *SIMULA 67*.

Jazyk *GPSS* připomínáme spíše z historických a pietních důvodů než pro praktickou využitelnost; jazyk *SIMSCRIPT* je typickým představitelem jazyků orientovaných

na samostatné události; konečně jazyk *SIMULA 67* je univerzální programovací jazyk, v němž je pomocí obecných prostředků kvaziparalelního programování definována třída *SIMULATION* jako problémově orientovaná část jazyka pro diskrétní simulaci.

5.2.2.1. *GPSS*

Vznik jazyka *GPSS* (*G*eneral *P*urpose *S*ystem *S*imulation) se datuje do roku 1961. Jeho princip je analogický jazyku *CSMP*, avšak s tím rozdílem, že je určen pro modelování diskrétních systémů, a to zejména systémů hromadné obsluhy.

Modelovaný systém se popisuje vývojovým diagramem (*flow diagram*), kterým procházejí požadavky na obsluhu, v jazyce GPSS nazývané *transakce*. Chování transakcí je dáno tvarem diagramu, který se skládá z různých typů bloků, z nichž namátkou jmenujme:

GENERATE	slouží pro generování transakcí se specifikovaným počtem parametrů ve specifikovaných časových intervalech,
TERMINATE	ukončuje průchod transakce systémem,
ADVANCE	modeluje spotřebu času na žádost transakce,
SEIZE	obsazuje některé obslužné místo — zařízení pro obsluhu transakcí; není-li obsluha volná, transkace čeká ve frontě,
RELEASE	uvolňuje obslužné zařízení získané pomocí *SEIZE*,
ASSIGN	mění hodnoty parametrů transakce,
SAVEVALUE	operuje nad globálními parametry modelu,
RN1, ..., *RN8*	generátory náhodných čísel.

Domníváme se, že již tento krátký výčet je dostatečně ilustrativní pro možnosti programátora při použití jazyka GPSS.

5.2.2.2. *SIMSCRIPT*

Jazyk *SIMSCRIPT* je vybudován na koncepci podprogramu událostí (*event routine*), což je nejjednodušší přístup k zobrazení událostí v programu. Při programování simulačního modelu v jazyce *SIMSCRIPT* musí být celá činnost modelovaného systému „rozdrobena" do jednotlivých typů událostí, z nichž každému odpovídá jeden podprogram. Neexistuje zde žádný příkaz modelující tok času (jako například *ADVANCE* v jazyce *GPSS*); jediný možný pohyb v čase záleží v naplánování další události pomocí plánovacího příkazu.

Podprogramy událostí reprezentují pouze stavové změny systému, je tedy ještě třeba reprezentovat stavy systému samotné a k tomu slouží *entity* jako obdoba transakcí v *GPSS*. Entita je datová struktura složená ze základních typů **real** a **integer**, která může být zařazena i do fronty. Speciálním typem entit jsou záznamy o události (*event notice*), které reprezentují položky simulačního kalendáře.

Program v jazyce *SIMSCRIPT* se skládá ze záhlaví, ve kterém jsou definovány

globální proměnné, entity a fronty; za ním následují příkazy formulující podprogramy událostí. Vytváření, respektive rušení entit, popřípadě záznamů o událostech se děje pomocí příkazů *CREATE*, respektive *DESTROY*.

Plánovací příkazy pro vyjádření vazeb mezi událostmi mají tvar:
CAUSE X AT Y , resp. *CANCEL X* ,

kde *X* je označení události prostřednictvím podprogramu události a Y je hodnota simulárního času.

Ostatní rysy jazyka jsou v podstatě shodné s jazykem FORTRAN.

5.2.2.3. *SIMULA 67*

Základním prostředkem programovacího jazyka *SIMULA 67* je tzv. třída (**class**), což je v podstatě programátorem definovaný prototyp datové struktury, která může mít nejenom údajové, ale i procedurální složky, tzv. lokální (též endogenní) procedury. Exempláře datové struktury podle prototypu definovaného třídou je možno v programu vytvářet pomocí tzv. generátoru **new**. Důležitou vlastností jazyka je tzv. prefixace, která umožňuje doplňovat definovaný prototyp struktury o další části.

K datové struktuře třídy může být též připojena operační část, která se začne provádět vždy po vytvoření nového exempláře. Mechanismem třídy tak mohou být v programu logicky spojeny stavové veličiny modelu (údajové složky exempláře třídy) s operacemi nad nimi prováděnými (endogenní procedury) a eventuálně též s dějem, který se odehrává po vstupu prvku do modelovaného systému (operační část třídy).

V rozšíření normy jazyka je možné chránit údajové složky exempláře struktury před zásahem z vnějška; tak je možné dosáhnout značně přehledného simulačního programu: stavové proměnné se mohou měnit jen prostřednictvím operátorů na nich lokálně definovaných.

Pro konstrukci diskrétních simulačních modelů je nejdůležitější vlastností jazyka *SIMULA 67* tzv. kvaziparalelní programování, jehož prostředky umožňují přerušit provádění operační části exempláře nějaké třídy a později se vrátit k provádění zbylého úseku přerušené operační části. Pomocí těchto prostředků je možno v jazyce realizovat předávání mezi jednotlivými simulačními procesy.

Programové prostředky pro simulaci diskrétních systémů jsou v jazyce *SIMULA 67* předdefinovány ve standardní třídě *SIMULATION* a jsou uživateli dostupné pomocí prefixace uživatelského programu nebo vhodného vnitřního bloku.

Vlastnosti procesu, kterým jsou popisovány děje v modelovaném systému, shrnuje lokální třída ve třídě *SIMULATION* deklarovaná jako:

link **class** *process*; ... ;

z prefixové třídy *link* získává vlastnosti potřebné pro vazbu do spojových seznamů (front). Ve třídě *process* jsou definovány procedury, které umožňují programátorovi zjistit momentální vnější stav procesu, resp. čas jeho příští aktivní práce. Tyto vlastnosti procesu jsou obvykle využívány prostřednictvím prefixu ke třídě deklarované uživatelem.

Simulační kalendář je ve třídě *SIMULATION* vytvořen jako spojový seznam *SQS* (sequencing set). Položkami tohoto seznamu jsou objekty třídy *EVENT NOTICE*, které uchovávají čas výskytu a ukazatel na proces, jehož plán reprezentují. Programátor má k seznamu *SQS* přístup pomocí procedur *current*, *time*, *hold*, *passivate*, *wait*, *cancel* a *ACTIVATE*.

Procedura *current* vrací odkaz na právě aktivní proces. Procedura *time* vrací okamžitou hodnotu simulárního času, tj. čas právě probíhající aktivní fáze procesu, který je current. Volání procedury *hold*, které je možné jen v právě aktivním procesu, znamená převedení tohoto procesu do suspendovaného stavu na dobu určenou parametrem procedury (přechod 1 —, resp. 0 — ve schématu na obr. 11). Stejným způsobem použitá procedura *passivate* nebo *wait* znamená převedení ze stavu aktivního do stavu pasívního (přechod 2 na obr. 11). Procedura *wait* navíc zařazuje proces na konec seznamu, který je určen jejím parametrem. Procedura *cancel* mění stav suspendovaného procesu na pasívní (přechod 5).

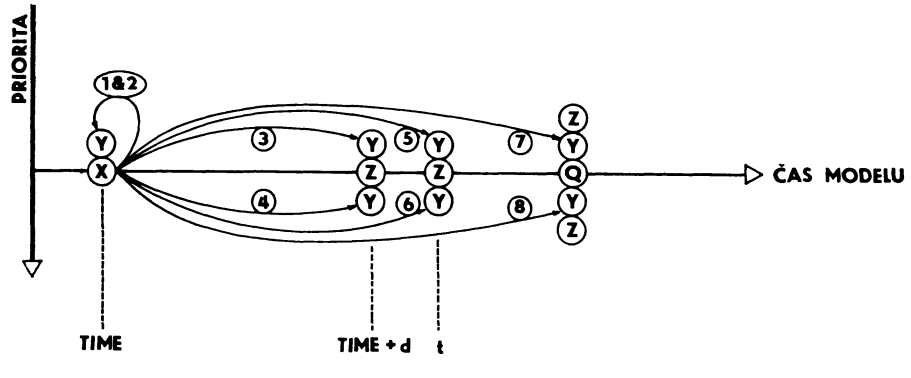
Obrácené přechody (t.j. 3, 4, 6 podle obr. 11) zprostředkuje procedura *ACTIVATE*, která je programátorovi nepřímo přístupná pomocí zvláštních plánovacích příkazů. Činnost osmi typů těchto příkazů s aktivátorem **activate** ukazuje obrázek 26. Stejné schéma platí i pro příkazy s aktivátorem **reactivate**, kde je navíc před vlastním plánováním zrušen plán procesu. existující eventuálně z dřívějška

Obr. 26.

X — plán právě aktivního procesu, v němž bylo použito **activate**,
Y — plán nově plánovaného procesu *Y*,
Z — plán procesu v *SQS* již dříve plánovaný,
Q — plán procesu *Q* (již dříve plánovaný v *SQS*),
PRIORITA — pořadí provádění v jednom okamžiku simulárního času,

Aktivační příkazy:

1 activate Y prior
2 activate Y
3 activate Y delay d prior
4 activate Y delay d
5 activate Y at t prior
6 activate Y at t
7 activate Y before Q
8 activate Y after Q



6. ZÁVĚR

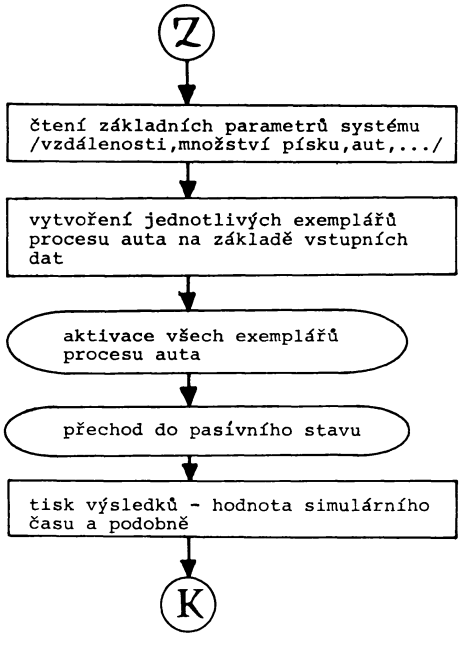
6.1. Ukázka konkrétní realizace simulačního programu

Ukažme si nyní na příkladu „Písek" faktický postup při realizaci simulačního programu.

Na sledovaném objektu jsme si nejprve vymezili modelovaný originální systém (například jsme se rozhodli, že budeme zanedbávat rozjíždění a brzdění automobilů, jejich popojíždění ve frontách a podobně). V paragrafu 2.1. jsme provedli základní analýzu tohoto originálního systému a podali dostatečně přesný slovní popis.

Vlastní postup při programování simulačního modelu závisí též dosti podstatně na tom, zda jej budeme budovat na procesovém principu nebo v samostatných událostech.

Použijeme-li procesovou realizaci, můžeme (jak jsme si ukázali v paragrafu 4.1.) seskupit jednotlivé události děje systému do procesů různým způsobem. Vyberme si např. realizaci děje jediným typem procesu auta. V programu pak budeme potřebovat dva typy procesů: jednak proces auta (popsaný v paragrafu 4.1.) a jednak proces zahajující a zakončující simulační výpočet. Programování v simulačních procesech je totiž vesměs realizováno tak, že jeden konkrétní proces v programu má význačné postavení: výpočet programu začíná jeho automatickým aktivováním v modelovém čase 0 (časový počátek) a při převedení tohoto procesu do ukončeného stavu končí též celý simulační výpočet. V našem případě může mít tento řídící proces tvar uvedený na obr. 27. Příkaz ukončení simulace uvedený ve schématu procesu auta na obr. 20 znamená aktivaci tohoto řídícího procesu.



Obr. 27

Uveďme ještě ukázku realizace simulačního programu pomocí zmíněných dvou procesů v jazyce *SIMULA 67*:

```
001  Simulation begin
002
003      process class auto (rychlost, nosnost, doba_nakladky, doba_vykladky);
004      real rychlost, nosnost, doba_nakladky, doba_vykladky;
005      begin
006
007          procedure jizda (vzdalenost); real vzdalenost;
008              hold (vzdalenost/rychlost);
009          procedure cekej (fronta); ref (head) fronta;
010              inspect fronta do
011              if empty then into (this head) else wait (this head);
012          procedure odjed (fronta); ref (head) fronta;
013              begin out; activate fronta.first end;
014
015          while true do
016          begin
017              cekej (fronta_nakladky);
018              if písek gt 0 then
019              begin
020                  pisek:= pisek — nosnost; nalozena:= nalozena +1;
021                  hold (doba_nakladky)
022              end
023              else passivate;
024              odjed (fronta_nakladky); jizda (ac); into (auta_v_cd);
025              inspect pred do
026                  if this link qua process.evtime gt time + cd/rychlost then
027                      reactivate this process after this link
028                  else jizda (cd)
029              otherwise jizda (cd);
030              out; jizda (db); cekej (fronta_vykladky);
031              hold (doba_vykladky); nalozena:= nalozena — 1;
032              if pisek le 0 and nalozena eq 0 then
033                  activate main;
034              odjed (fronta_vykladky); jizda (ba)
035          end
036      end auto;
```

```
037    ref (head) fronta_nakladky, fronta_vykladky, auta_v_cd;
038    real pisek, ac, cd, db, ba; integer nalozena;
039
040    fronta nakladky:— new head; fronta vykladky:— new head; auta_v_cd: — new head;
041    pisek:= inreal;
042    ac:= inreal; cd:= inreal; db:= inreal; ba:= inreal;
043
044    while not lastitem do
045        activate new auto (inreal, inreal, inreal, inreal) at time;
046
047    passivate;
048
049    outtext ("hromada odvezena na misto ""B""" v case:"); outfix (time, 5, 10); outimage
050
051 end program
```

(Program řeší otázku, za jak dlouho bude hromada odvezena známou skupinou nákladních aut.

Použijeme-li realizaci programu v samostatných událostech, musíme na rozdíl od procesové realizace zobrazit v programu též stavy modelovaného systému (v procesovém pojetí jsou reprezentovány vnitřními stavy procesů).

Můžeme např. pro každé auto založit exemplář datové struktury s polem logických hodnot (či proměnnou skalárního typu a podobně), které bude určovat, v jakém stavu se dané auto nachází („čekání ve frontě na nakládání", „nakládání", „jízda z „**A**" do „**C**" " a podobně).

Pro každý jednotlivý typ události (začátek nakládky, konec nakládky, ...) budeme deklarovat proceduru s parametrem typu auto (nebo lépe lokální proceduru ve struktuře auto). V simulačním kalendáři (patrně prostého nehierarchického typu) pak můžeme mít plány obsahující pouze čas a odkaz na strukturu auta, kterého se událost týká. Podle stavu, ve kterém se auto nachází, snadno poznáme, jakou proceduru (tj. jakou událost) na něj máme aplikovat.

Zahájení a ukončení simulace se bude odehrávat velmi podobně jako v procesové realizaci.

6.2. Aplikace

Závěrem celého článku bude jistě účelné se zmínit o možnostech použití simulačních metod pro řešení praktických problémů a dotknout se v krátkosti několika aplikací.

Především — všeobecně řečeno — modelování je možné použít pro všechny systémy,

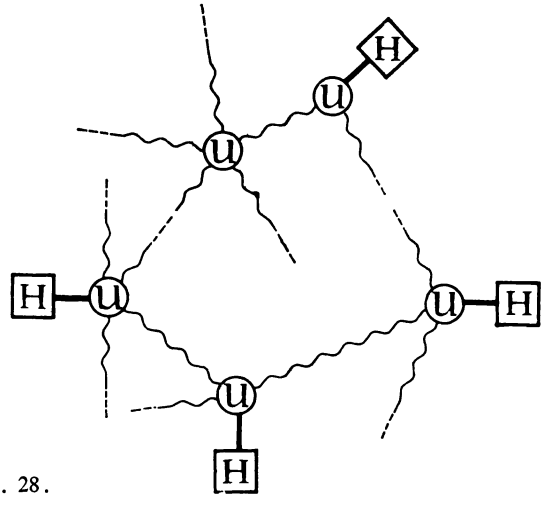
u kterých jsme schopni dostatečně jasným způsobem popsat jejich chování. Homomorfismus modelového stavového zobrazení přitom ještě dovoluje různou úroveň abstrakce. (Samozřejmě jinou otázkou zůstává, kdy je použití modelu vhodné nebo se vyplácí. Podobně je jinou otázkou, zda jsme schopni aplikovat výsledky simulace na modelovaný originál a zda má z tohoto důvodu smysl něco modelovat či ne.)

Autorům je známa řada simulačních projektů, které byly v Československu úspěšně použity (samozřejmě je jim známa i řada projektů, o kterých to prohlásit nelze). Z těch úspěšných uvádíme namátkou několik s jejich stručnými charakteristikami.

6.2.1. Počítačová síť [1]

Systémy počítačových sítí jsou modelovány na nějaké zvolené úrovni zobrazení technických a programových prostředků za účelem získání výhodných charakteristik, respektive obráceně za účelem dostatečného dimenzování při zadaném výkonu, odhalení úzkých profilů a podobně.

Vzhledem k tomu, že modely těchto systémů jsou typickými úlohami diskrétní simulace (a ukázka doplňuje paragraf 5.1. našeho článku), uvádíme zde podrobněji z řady existujících programů model řízení toku dat v počítačové síti.



Obr. 28.

Počítačová síť je tvořena množinou uzlových počítačů („**U**" na obr. 28), z nichž libovolné dva mohou být propojeny spojem přenosu dat. Ke každému uzlu může být připojen nejvýše jeden hlavní počítač („**H**" na obr.), který je zdrojem i příjemcem dat v síti.

Komunikace mezi hlavními počítači probíhá vždy ve dvojicích zvaných logické kanály. Jeden z hlavních počítačů v logickém kanále vysílá zprávu složenou z několika tzv. paketů do koncového počítače tohoto kanálu, který potvrzuje příjem zprávy vysláním speciálního paketu zpět. Cesta paketů z počátečního do koncového počítače není dána pevně, závisí na stavu sítě a může se v průběhu činnosti celého systému měnit.

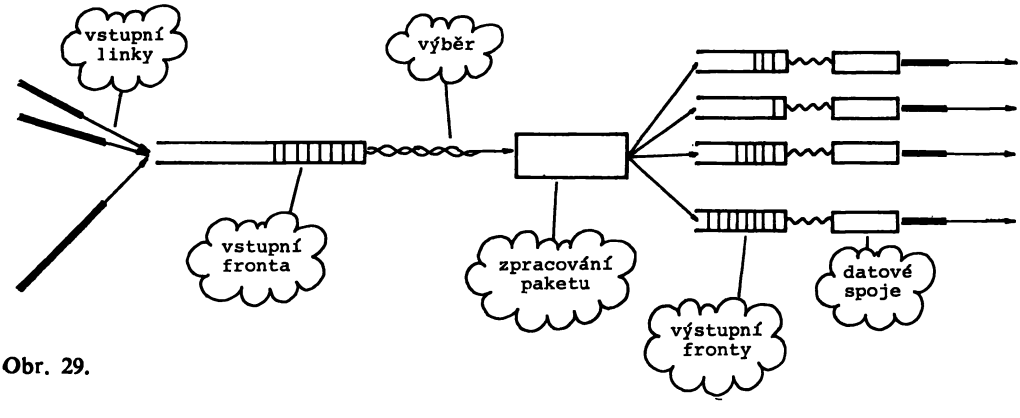
Každý uzlový počítač představuje systém hromadné obsluhy podle obr. 29. Řízení uzlu probíhá ve dvou úrovních:

– příjem a odeslání paketu po datových spojích,
– funkce směrování a řízení toku.

Zpracování paketu na uzlovém počítači znamená určení dalšího směru jeho postupu na cestě do cílového počítače, a tedy zařazení do některé z výstupních front. Směr postupu je dán cílovou adresou (ta je v paketu obsažena) a momentálním stavem sítě, tj. délkou front v uzlu, rychlostí spojů, délkou cesty a podobně. Řazení a výběr z front může být prioritní podle typu paketu. Uzlový počítač ještě sleduje vytížení všech směrů sítě a v případě, že délka některé jeho fronty překročí stanovenou mez, vysílá do zdroje v příslušném logickém kanále tzv. škrticí paket, který zdrojovému počítači signalizuje, že má o jistou hodnotu omezit tok informace.

Vstupními parametry modelu je počet uzlových a hlavních počítačů, rychlosti spojů, počet vyrovnávacích pamětí v uzlech a doby zpracování, směrovací tabulky uzlů a některé další parametry pro globální řízení toku dat.

Na výstupu modelu je možné detailně sledovat chování systému v zadaných okamžicích simulárního času. Po ukončení činnosti poskytuje model souhrnné charakteristiky jako celkové množství přepravených zpráv, využití spojů a vyrovnávacích pamětí, délky front, doby přenosu zpráv na logických kanálech a další.



Obr. 29.

6.2.2. Kompartmentová analýza [2, 10]

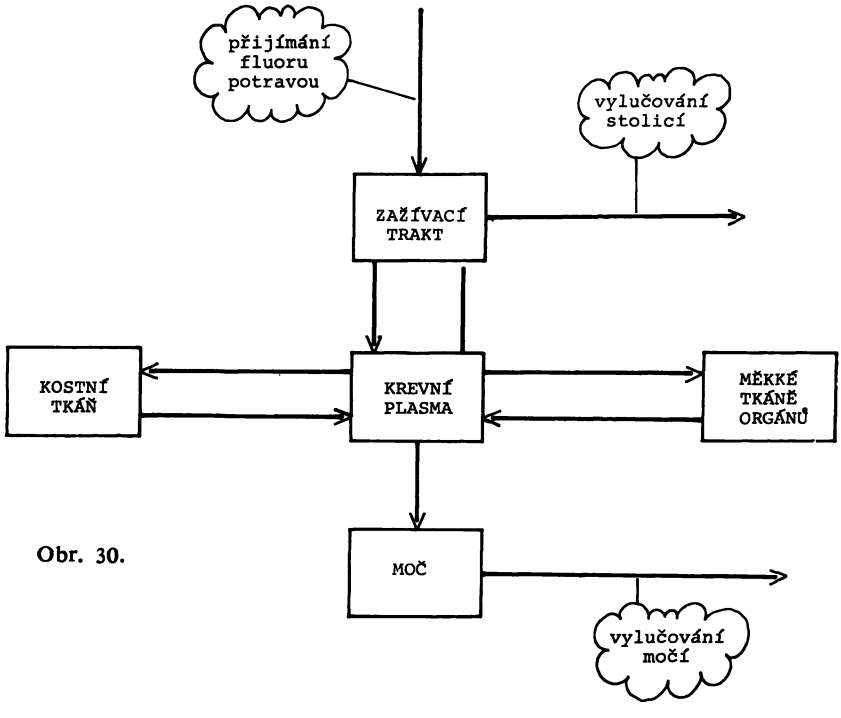
Z větší řady simulačních projektů z biologie a medicíny se zmíníme o modelování metabolických pochodů, které je založeno na tzv. kompartmentové analýze.

Kompartmentem (nebo též poolem) rozumíme ohraničenou část metabolického systému, která obsahuje určitou sledovanou látku ve všech místech ve stejné koncentraci. Transport sledované látky z jednoho poolu do druhého je dán transportní konstantou, která určuje poměrnou část látky v poolu, a přejde do druhého poolu za jednotku času. (Transportem zde rozumíme jak prostý transport bez chemických změn, tak i přestup látky s biochemickou transformací.)

Pro jednotlivé případy metabolických pochodů je možno sestavit schéma uvažovaných

kompartmentů a transportu látek mezi nimi. Například pro pochody ukládání fluóru v kostech lidského organismu může mít kompartmentové schéma tvar podle obr. 30.

Samozřejmě se zde dopouštíme jistého zjednodušení. Uvažujeme totiž, že kdykoli vstoupí nějaká látka do poolu, okamžitě se v něm rozptýlí tak, že má ve všech jeho místech stejnou koncentraci. Porovnání modelových výsledků s přímými laboratorními pokusy však ukazuje, že takové zjednodušení není příliš na závadu.



Obr. 30.

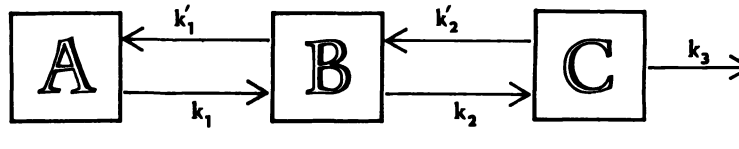
Kompartmentové systémy jsou typickými případy spojitě modelovaných systémů. Například systém vyplavování látky z kompartmentu „**A**" přes pooly „**B**" a „**C**" podle obr. 31 lze popsat soustavou diferenciálních rovnic:

$$dm_A/dt = k_1' m_B - k_1 m_A$$

$$dm_B/dt = k_1 m_A + k_2' m_C - (k_1' + k_2) m_B$$

$$dm_C/dt = k_2 m_B - (k_2' + k_3) m_C ,$$

kde m_J označuje množství sledované látky v kompartmentu „**J**".



Obr. 31.

Jak je patrné, přepis kompartmentového schématu do soustavy diferenciálních rovnic je do značné míry mechanická záležitost. Proto byly také vypracovány programovací

prostředky pro modelování kompartmentových systémů pouhým přepisem kompartmentového schématu podle jistých syntaktických pravidel do textového tvaru programu (např. jazyk *COSMO* – viz. [13]).

Složitější situace kompartmentových systémů, např. s dynamickým počtem poolů, dynamickým počtem transformačních cest, nekonstantními tokovými rychlostmi a podobně, jsou řešeny modely kombinovaného typu.

Pomocí kompartmentové analýzy byla řešena celá řada úspěšných simulačních projektů, jako model metabolismu jódu ve štítné žláze, modely vylučování léčiv z organismu, modely glukózového metabolismu a podobně.

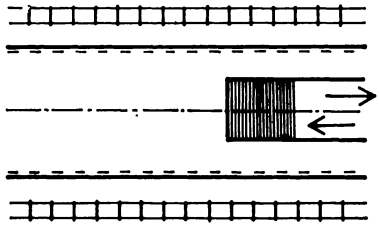
Poznamenejme ještě, že kompartmentová analýza se nepoužívá jen k metabolickým studiím. Je možno ji použít i pro jiné biologické systémy (například v ekologických studiích).

O dalších třech projektech se zmíníme již jen v krátkosti.

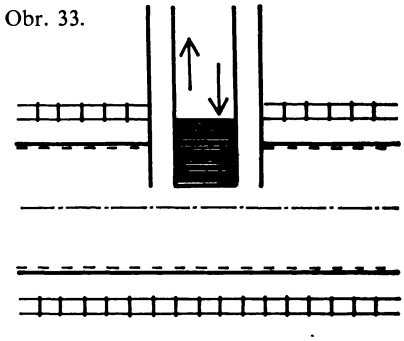
6.2.3. Nástupiště v metru

Šlo o modelování stanice metra za účelem rozhodnutí, zda má být schodiště umístěno tak, aby jeho dlouhá osa byla v ose nástupiště (obr. 32), nebo naopak kolmá na osu nástupiště (obr. 33).

Obr. 32.



Obr. 33.



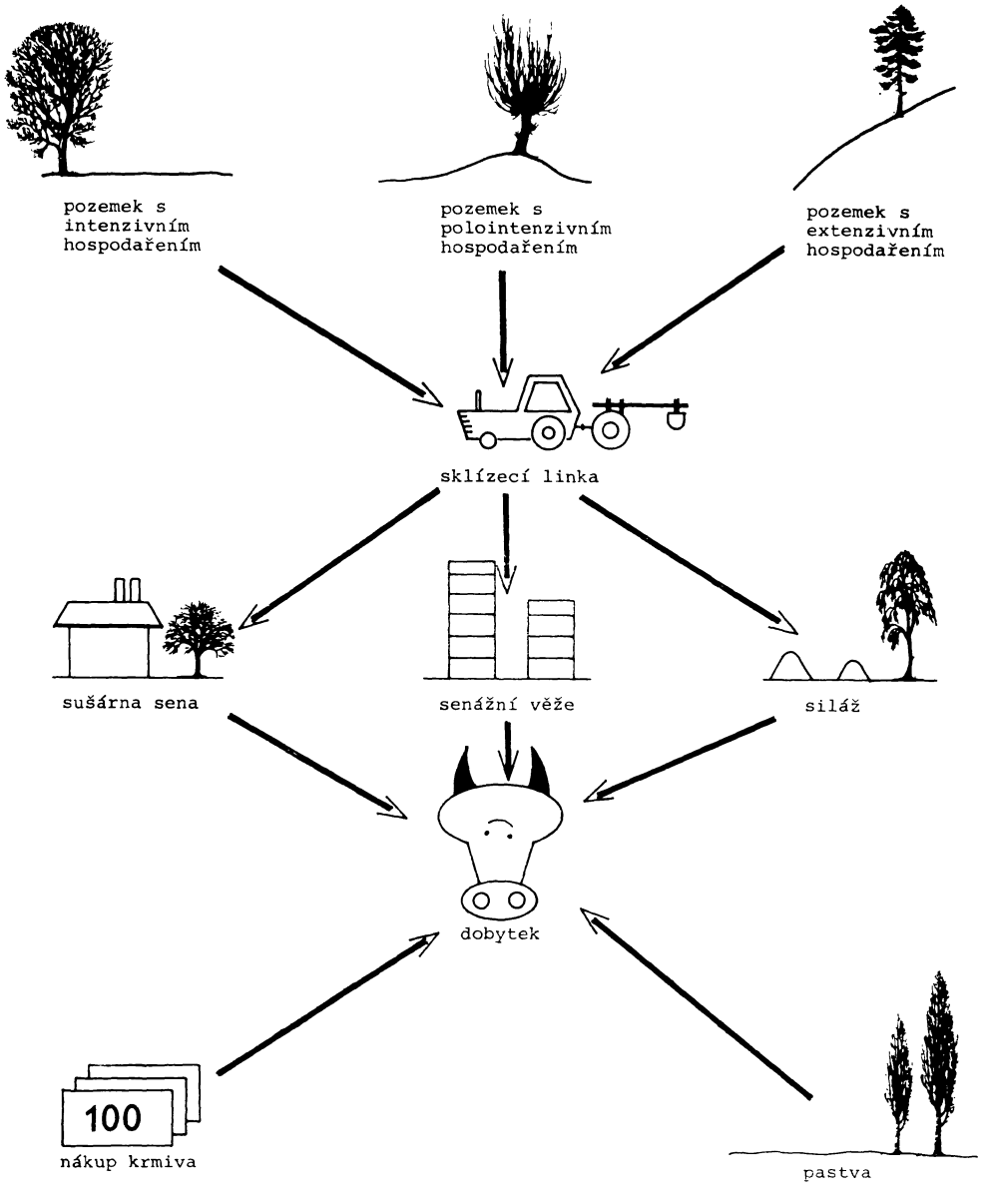
Byl modelován pohyb cestujících po nástupišti a schodišti a sledovala se propustnost a vytváření front na úpatí schodiště v obou uspořádáních.

6.2.4. Studie „Želivka" [11, 16]

Pro potřeby státního statku v Chotěboři byla vypracována rozsáhlá simulační studie za účelem nalézt optimální počet „dobytčích jednotek", pro které by bylo možno (s jistými přípustnými ztrátami) získávat krmivo na předem známých pozemcích různě výnosných typů.

V modelu bylo třeba brát ohled na náhodné změny počasí (v realizačním programu pseudonáhodně generované) a na celou řadu anomálních omezujících podmínek. (Např. část píce, která na pozemcích uzrála, se přednostně zpracovává pomocí sušárny

sena, ale tu si státní statek pronajímá a může si ji pronajmout pouze na pracovní dny každého sudého týdne.)



Obr. 34.

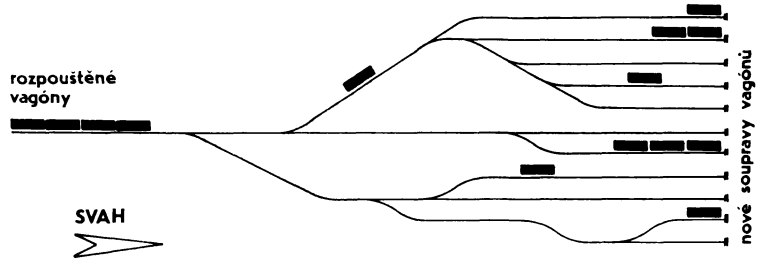
V podstatě šlo o simulaci diskrétního systému materiálového toku krmiva s velice složitými frontovými režimy podle schématu na obr. 34.

6.2.5. Seřaďovací nádraží [9]

Na katedře technické kybernetiky v dopravě VŠDS Žilina byla vypracována simulační studie seřaďovacího nádraží za účelem zjištění optimálního tvaru z různých stavebně možných variant (obr. 35).

Sledovány při tom byly zejména tyto parametry:

- průběh jízdy jednotlivých vozů,
- průměrná doba rozpouštění jedné soupravy,
- počet souprav a vozů rozpuštěných za jednu pracovní směnu,
- průměrná délka mezery mezi stojícími vozy,
- počet nedodržení náběhové rychlosti.



Obr. 35.

Literatura

[1] Benda Z.: *Možnosti použití simulačních modelů v návrhu počítačových sítí.* Výzkumná zpráva úkolu SPEV-V-02/12-309-03. VUT Brno, 1980.

[2] Carson E., Jones A.: *Use of Kinetic Analysis and Mathematical Modeling* in the *Study of Metabolic Pathways in Vivo.* New Engl. Jour. of Med., *300*, 1979, May 3 a May 10, str. 1016—1027 a 1078—1086.

[3] Černý P., Kotva M.: *Dohoda o obsahu pojmu „Simulace systémů".* Automatizace, 1978; 21.

[4] Dahl O. J., Myhrhaug B., Nygaard K.: *SIMULA 67 Common base language.* NCC, Oslo, 1970.

[5] Douša J.: *Koncepce třídy SCDP.* Sborník Moderní programovací jazyky, 1979, str. 120—130.

[6] Douša J.: *Synchronizace simulací spojitých procesů v kombinovaném simulačním systému.* Sborník MOP '80, 2. díl, 1980, str. 31—39.

[7] Gruska J., Wiedermann J., Černý A.: *Typy a štruktúry dát.* Sborník SOFSEM '78, 1978.

[8] Hobza V.: *Systém pro realizaci čekání simulačních procesů na podmínku.* Diplomová práce, MFF UK, Praha, 1982.

[9] Hora P.: *Simulačný model prevádzky spádoviska.* Sborník SIMULA 67, 1968, str. 98—106.

[10] Horová M.: *Kombinovaná simulace kompartmentových systémů.* Diplomová práce, MFF UK, Praha, 1979

[11] Chochol Š. a kol.: *Simulace výroby, konzervace a spotřeby pícnin v podniku.* Zeměd. Techn., *27*, 1981, str. 613—622.

[12] Kindler E.: *Dynamic Systems and Theory of Simulation.* Kybernetika, *15*, 1979, 2, str. 77-87.

[13] Kindler E.: *Simulační programovací jazyky.* SNTL, Praha, 1980.

[14] Klír J., Valach M.: *Kybernetické modelování.* SNTL, Praha, 1965.

[15] Malík M.: *Simulace elektrontransportních reakcí fotosyntetického řetězce.* Disertační práce, MFF UK, Praha, 1978.

[16] MALÍK M.: *Simulační studie „ŽELIVKA"*. Sborník Moderní programovací jazyky, 1979, str. 52—62.
[17] PYTJEVA N. a KOL.: *O funkcionalnoj organizacii elektrontransportnoj cepi v chromatoforach fotosintezirujiščich bekterij*. Studia Biophysica, *48*, 1975, 3, str. 173—184.
[18] RØGEBERG T.: *Simulation and Simulation Languages*. NCC, Oslo, 1973.
[19] SERBA I.: *Modelování na počítačích I.* Učební texty vysokých škol, SNTL, Praha, 1974.
[20] SERBA I.: soukromé sdělení, 1980.
[21] ZEIGLER B.: *Theory of Modelling and Simulation*. John Wiley and Sons, New York, 1976.
[22] ZÍTEK F.: *Ztracený čas*, Academia, Praha, 1969.
[23] *System/360 — Continuous-System Modeling Program*. IBM User's Manual, IBM, 1967.
[24] *GPSS/360*. IBM User's Manual, IBM, 1964.
[25] *SIMSCRIPT 1.5 IBM System 360/370*. IBM User's Manual, IBM, 1971.
[26] *Problem Statement Language*. ISDOS Manual, Univ. of Michigan, Michigan, 1980.
[27] *Problem Statement Analyzer*. ISDOS Manual, Univ. of Michigan, Michigan, 1979.

Vznik sluneční soustavy

Zdeněk Pokorný, Brno

1. Úvod

Před necelými 5 miliardami let došlo k důležité události — vznikla sluneční soustava. Zrekonstruovat situaci a popsat vznik naší planetární soustavy je neobyčejně obtížné, a to nejen pro časovou odlehlost; dodnes se totiž nepodařilo objevit analogickou soustavu u některé jiné hvězdy. Zjištění přítomnosti planet u hvězdy je z pozorovatelského hlediska mnohem složitější než např. pozorování zárodečné mlhoviny, obklopující mladou hvězdu. Není vyloučeno, že mezi už známými kosmickými zdroji infračerveného záření se nacházejí i zárodečné mlhoviny, v nichž vznikají nové planety.

Pohled do historie ukazuje, že otázkami vzniku sluneční soustavy se zabývala řada vynikajících vědců. Nebylo by bez zajímavosti jistě ani dnes sledovat myšlenkové postupy těchto badatelů a soupeření jednotlivých koncepcí, založených ovšem více na spekulacích a odhadech než na reálných datech. Naším úkolem je však podat současný pohled na věc, a proto čtenáře, který hledá informace o historii oboru, odkazujeme na literaturu [1].

Vznik planet úzce souvisí se vznikem hvězd. Podle dnes všeobecně přijímané koncepce vzniku sluneční soustavy se Slunce i planety utvořily téměř současně. V důsledku kolapsu části oblaku mezihvězdné látky, ke kterému došlo před asi $(4{,}6-4{,}7).10^9$ lety, se zformovala zárodečná mlhovina. Pak, v časové škále nepřevyšující $(1-4).10^8$ let, vzniklo Slunce, planety i množství „drobných" těles.