Dependabilita informačních systémů

Opora DEP

KI/DEP

Doc. Viktor Maškov DrSc.

Ústí nad Labem 2020

Studijní materiál byl vytvořen v rámci projektu Univerzita 21. století – Kvalitní, moderní a otevřená instituce, reg. č. CZ.02.2.69/0.0/0.0/16_015/0002408 (KA02 Podpora a rozvoj polytechnických studijních programů).

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

OBSAH

Úvodní slovo

Kurz uvádí do problematiky dependability informačních systémů. V rámci kurzu budou podrobně vysvětleny otázky samokontroly a samodiagnostiky počítačových systémů.
Kurz bude zaměřen zejména na spolehlivost a odolnost informačních systémů proti závadám.
Studium se skládá ze dvou částí. Jedna část (distanční) spočívá v samostatné práci studentů. Druhá část (kontaktní) probíhá ve formě seminářů a konzultací. V průběhu samostatné práce studenti musí splnit úkoly seminářů, které jsou uvedeny v kapitolách. Pro správné řešení úkolů studenti musí prostudovat doporučenou literaturu a obsah nabízených prezentací (jsou uvedeny v každé kapitole).
V průběhu seminářů a konzultací studenti musí ukázat výsledky úkolů. Zároveň studenti budou mít možnost položit otázky a dostat vysvětlení problémů, které nezvládli pochopit.
Kurz bude ukončen zápočtem a navazující zkouškou. Pro získání zápočtu student musí doložit výsledky všech úkolů. Přitom 80% úkolů musí být řešeny správně. Po získání zápočtu student může se přihlásit ke zkoušce.
Na zkoušce studen vylosuje jednu otázku ze předem připraveného seznamu otázek. Otázky jsou vymezeny obsahem kurzu a specifikují úkoly kapitol. Seznam otázek bude zveřejněn dva týdny před zkouškou. Při hodnocení odpovědi studenta na otázku má prioritu schopnost studenta prakticky využít svoje znalosti.

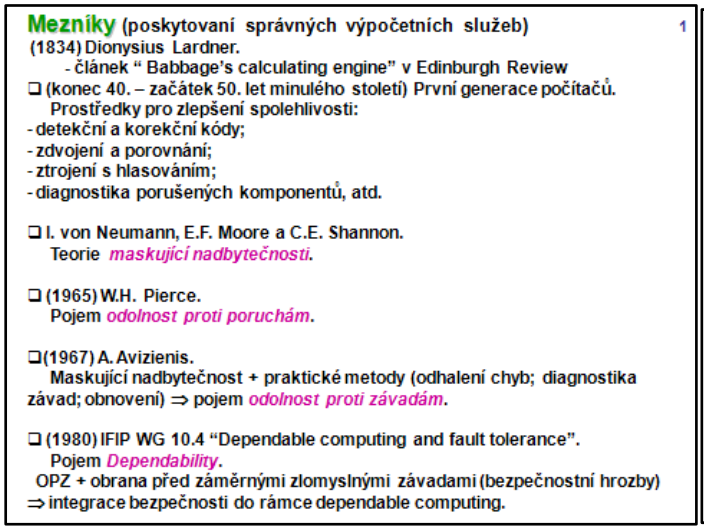
Kapitola 1. Úvod do problematiky

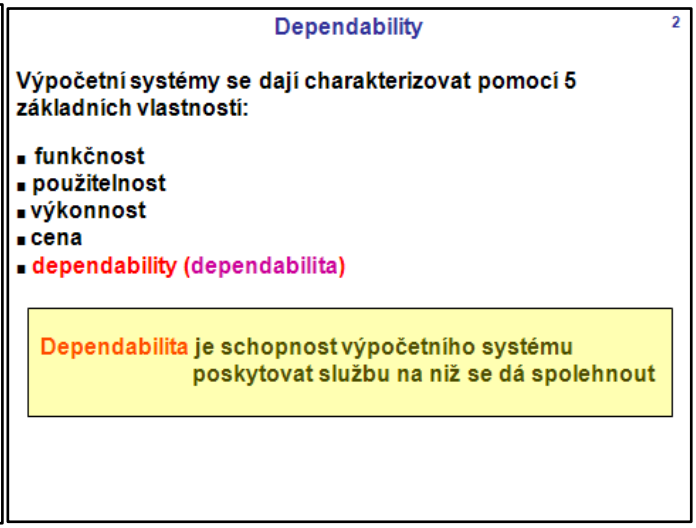
Cíl kapitoly:

- vysvětlit vznik a význam pojmu a koncepce „Dependabilita“
- charakterizovat základní prvky Dependability informačních systémů

Klíčová slova: dependabilita, atributy dependability, služba, selhání, docílení dependability

Výkladová část:
Následující slajdy vysvětlují hlavní cíly kapitoly



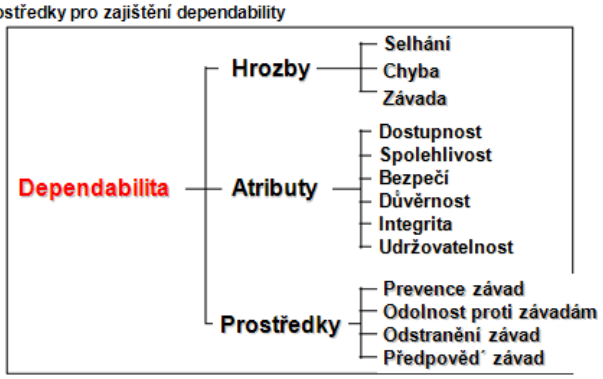


Dependability

3

Tři důležité části při realizaci ***Dependability*** :

- ❑ Hrozby
- ❑ Atributy
- ❑ Prostředky pro zajištění dependability



Dependability

4

Hrozby

Závada – je posouzená nebo hypotetická příčina chyby.

↓

Chyba (chybný stav) – je takový stav systému, který může vést k selhání.

↓

Selhání (porucha) – je událost, která se vyskytuje když se poskytována služba liší od spravné služby.

◘ Systém může selhat bud´ protože se nedodrží specifikace, nebo specifikace nepopisuje adekvátně funkce systému.

Režimy selhání (způsoby, jakým systém může selhat)

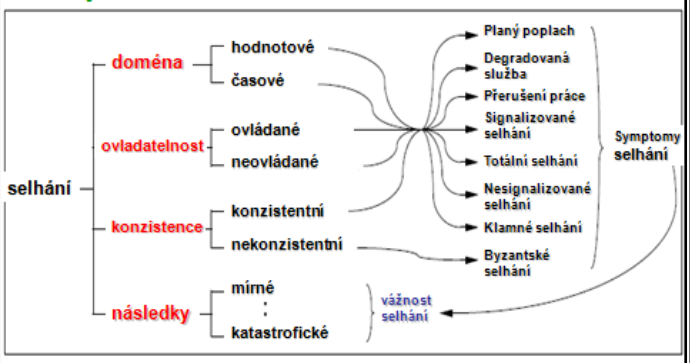
Podle 4 různých hledisek režimy mohou být charakterizovaný takto:

- selhávací doména;
- ovladatelnost selhání;
- konzistence selhání, když systém má dva a více uživatelů;
- následky selhání pro prostředí.

Dependability

5

Hrozby



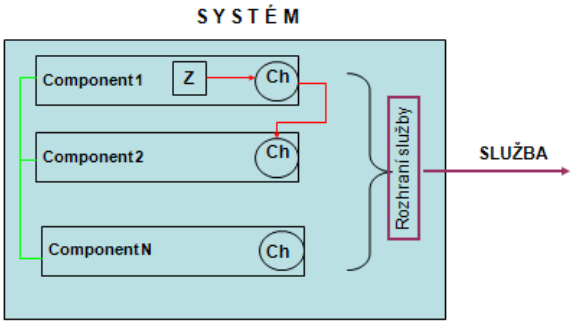
Obrázek ukazuje režimy selhání a také ukazuje možné symptomy selhání, které jsou výsledky kombinací: domény, ovladatelností a konzistence. Symptomy selhání mohou být mapovány na závažnost selhání, které jsou třídění následků selhání.

Dependability

6

Hrozby

Závada (Z) obvykle způsobí **Chybu (Ch)** ve stavu jednoho nebo více komponentů, ale k **Selhání** systému nedojde do té doby pokud **Chyba** nedosáhne **Rozhraní služby** systému.



Dependability

7

Hrozby

Třídění chyb

- ❑ Hodnotové ver. Časové chyby
- ❑ Konzistentní ver. Nekonzistentní (“Byzantské“) chyby
- ❑ Chyby různých závažností:
  méně závažné ver. obvyklé ver. katastrofické

■ Chyba je odhalená, jestli na její přítomnost ukazuje zpráva nebo signál.

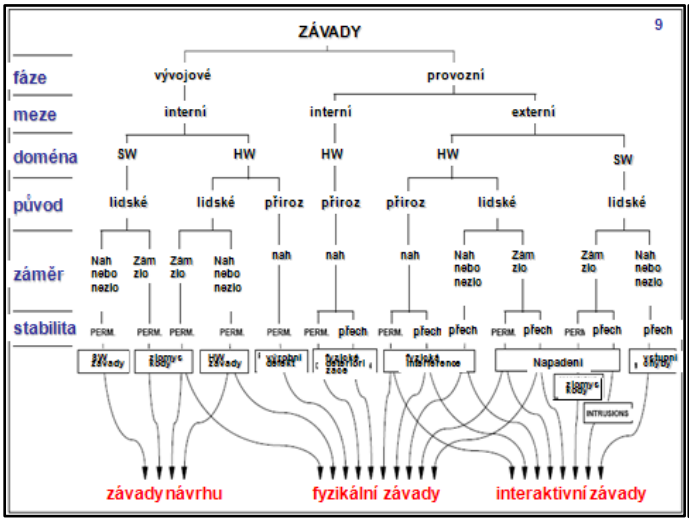
■ Chyba která je přítomná, ale neodhalená se jmenuje Latentní chyba.

Dependability

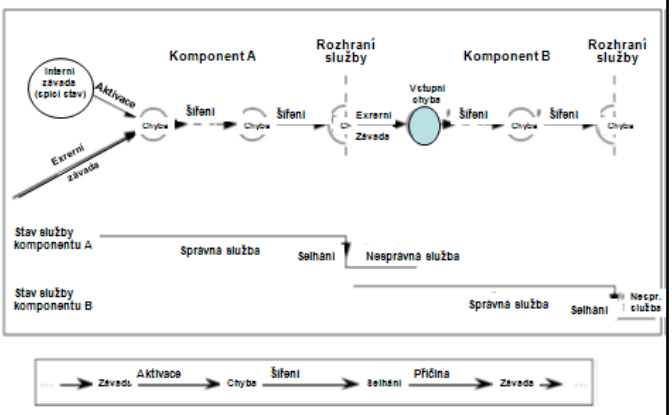
8

Tři hlavní třídy závad:

- závady návrhu;
- fyzikální závady;
- interaktivní závady.



Dependability
Vztah mezi Závadami, Chybami a Selháními



Dependability

Závady podle jejich *aktivační reprodukovanosti*

ZÁVADY
- **Stálé (pevné)**
- **Nestále (občasné) nebo unikající**

Terminologie pro SW:
- Unikající závady návrhu (bugs) = **Heisenbugs;**
- Stálé závady návrhu = **Bohrbugs.**

	Výpočetní systémy (např. [Gray 1990], [Cramp et al. 1992])		Řízené systémy (např. Komerční letadla [Ruegger 1990], telefonní síť [Kuhn 1997])	
	Hodnocení	Procento selhání	Hodnocení	Procento selhání
Interní fyzické závady	3	~10%	2	15-20%
Externí fyzické závady	3	~10%	2	15-20%
Interaktivní závady způsobené člověkem	2	~20%	1	40-50%
Závady návrhu SW	1	~60%	2	15-20%

Dependability

Atributy Dependability
- ❑ **Dostupnost:** pohotovost k provedení správné služby.
- ❑ **Spolehlivost:** kontinuita poskytování správné služby.
- ❑ **Zabezpečenost:** absence katastrofických následků pro uživatele a prostředí.
- ❑ **Důvěrnost:** absence nepovolených odhalení informace.
- ❑ **Integrita:** absence nevhodných změn stavu systému.
- ❑ **Udržovatelnost:** schopnost procházet opravy a modifikace.

Jiné atributy dependability:
Robustnost – odolnost proti chybným vstupním datům.

Důvěrnost
Integrita
Dostupnost
} **Bezpečnost** (absence neoprávněného přístupu ke stavu systému nebo neoprávněného ošetření stavu systému)

Sekundární atributy:
- zodpovědnost;
- originalita;
- nepopíratelnost

Dependability

Běžné otázky:
- Proč SW systémy ***selhávají***?
- Co tvoří základ ***procesu selhání*** SW?
- Jestli-že selhání SW jsou "***systematické***", proč pořad mluvíme o spolehlivosti s použitím termínů pravděpodobnosti?

Systematické: pokud se závada tohoto druhu (systematická závada) projeví v určitých podmínkách, pak můžeme garantovat že tato závada se projeví vždycky při opakování podmínky.

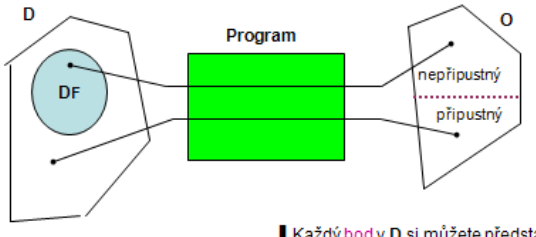
Selhání SW jsou vždy výsledkem *závad návrhu*, které se projeví při určitých výpočetních okolnostech. Tyto závady (závady návrhu = bugs) budou v SW od okamžiku jeho vytvoření, nebo při následujících změnách.

Proces selhání – je proces ve kterém se závady jeví jako výsledek zpracování (výpočtu) vstupních dat.

Dependability

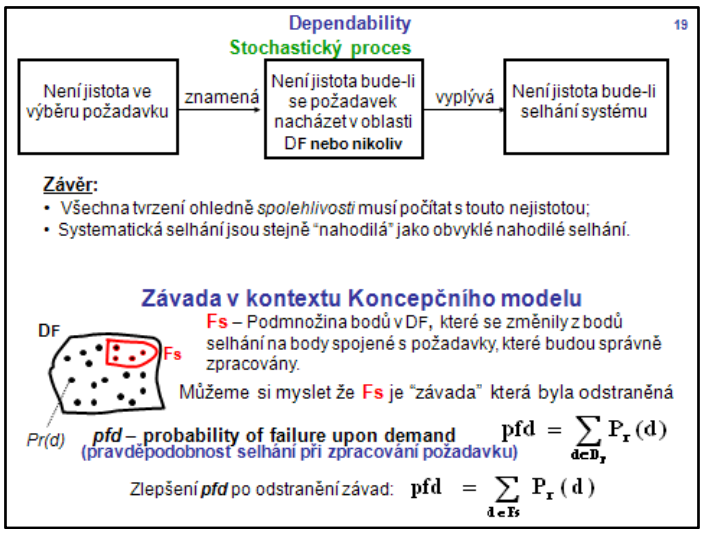
Koncepční model procesu selhání SW

Systém na vyžádání (demand-based system)
(např. Systém zajišťující bezpečnost jaderné elektrárny)



D – prostor požadavků;
O – množina výstupních dat;
DF – množina požadavků, které způsobí selhání systému;
● - konkrétní fyzický požadavek.

Každý bod v **D** si můžete představit jako konkrétní fyzický požadavek. (např., vektor teplot, tlak, rychlost toku apod., odebrané v pravidelných intervalech skenovacími senzory)



Kontrolní otázky:

- vysvětlete základní prvky dependability
- charakterizujte atributy dependability
- definujte pojmy: závada, chyba a selhání

Úkoly pro samostatnou práci:
1) Prostudovat atributy dependability a prostředky pro docílení dependability informačních systémů. Použit literaturu [1], [2] a [3].
2) Vytvořit tabulku s příklady selhání různých systémů. Pro každý příklad popsat závadu, chybu a selhání systému. Tabulka má mít 10 řádků (jeden řádek pro každý příklad) a 4 sloupce. První sloupec pro popis systému, druhý pro popis závady, třetí sloupec pro popis chyby a čtvrtý sloupec pro popis selhání systému.

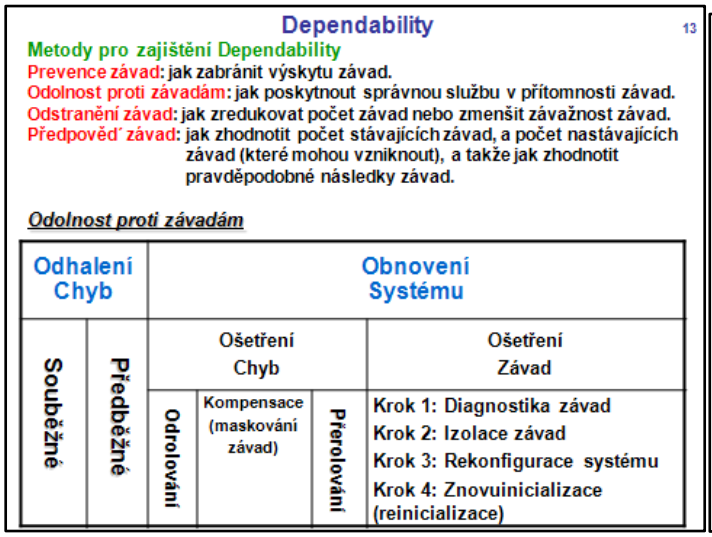
Kapitola 2. Odolnost informačních systémů proti závadám

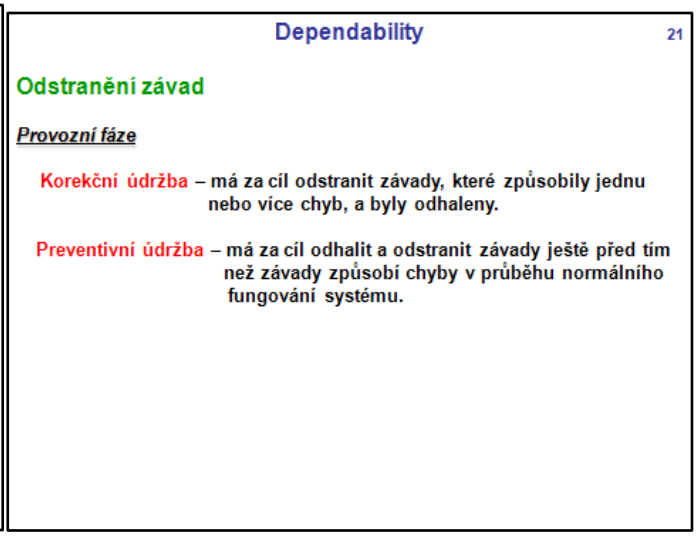
Cíl kapitoly:

- vysvětlit koncepci odolnosti informačních systémů proti závadám
- podrobně rozebrat otázky odhalení chyb následujícího obnovení systému
-

Klíčová slova: odolnost proti závadám, chybný stav, selhání systému, odhalení chyb

Výkladová část:
Následující prezentace vysvětluje hlavní cíly kapitoly





Odstranění závad

Vývojová fáze	Provozní fáze
1. Verifikace & Validace 2. Diagnostika 3. Korekce	Korekční údržba Preventivní údržba

Verifikace- je proces kontroly jestliže systém dodržuje nastavené vlastností.
Validace – je proces kontroly specifikace systému.

Cena V&V = ½ nebo ¾ ceny vývoje systému

V&V umožňuje: zredukovat četnost závad

z
10 – 300 závad/kLoC → následek vývoje SW
na
0.01 – 10 závad/kLoC → po odstranění závad

Důležité: 1 závada/kLoC zůstává v systému (v průměru)

Předpověď závad

Striktní odvozovací procedura:

- Systém na vyžádání (e.g. Ochranný systém)
  Když potřebujeme 99% jistoty že pfd nebude horší než 10^{-3}, musíme obdržet přibližně 4600 požadavků, které by nevedly k selhání;
  Pro 99% jistoty v 10^{-4}, toto číslo stoupá na 46000 požadavků.
  (Toto testování bylo provedené pro ochranný systém jaderného reaktoru SizeWellB v UK).
- Systém s nepřetržitou obsluhou (e.g. Řídicí systém)
  99% jistoty v MTTF (střední doba do poruchy) 10^4 hodin (1.14 roku) potřebují přibližně 46000 hodin testování bez selhání (t.j. Bezporuchového testování);
  Zvýšení MTTF na 10^5 hodin, bude potřebovat dobu testování 460000 hodin.

Efektní pravděpodobnostní metody

Krátkodobé pozorování › předpověď na dlouhé období

CRL:
"krátkodobé pozorování přidává velmi málo k jistotě že systém bude dlouho bezporuchové fungovat v budoucnosti".

Odolnost proti závadám

Odhalení chyb

Odhalení chyb: se projevuje jako chybový signál nebo zpráva o chybě uvnitř systému.
Souběžné OCh – probíhá v průběhu poskytování služby;
Předběžné OCh – probíhá když poskytování služby je pozastaveno;

Obnovení systému

Obnovení systému: transformuje stav systému, který obsahuje jednu nebo více chyb a závad, do stavu bez odhalených chyb a závad, které by mohly být znovu aktivovány.

A. ***Ošetření chyb***: odstraňuje chybu ze stavu systému.
- Odrolování,
  když transformace stavu probíhá jako vracení stavu systému do zachovaného stavu, který byl před odhalením chyby;
- Kompensace,
  když chybový stav obsahuje dostatek nadbytečnosti pro odstranění chyby;
- Přerolování,
  přechod do nového stavu, který neobsahuje již odhalené chyby.

Odolnost proti závadám

B. ***Ošetření závad***: zabraňuje lokalizovaným závadám v opakované aktivaci.

1) Diagnostika závad
   Identifikuje a zaznamenává příčiny chyb. V záznamech se uvádí lokalita a typ závady;

2) Izolace závad
   Vykonává fyzické nebo logické vyloučení závadných komponentů z další účasti v poskytování služby;

3) Rekonfigurace systému
   Buď přepíná na náhradní komponenty nebo používá nezávadné komponenty (které zůstaly v systému) a přerozdělí úkoly mezi nezávadnými komponenty;

4) Znovuinicializace
   kontrola, aktualizace a záznam nové konfigurace.

Odolnost proti závadám

Systémy s ovladatelným selháním: systémy jejichž návrh a implementace jsou takové že tyto systémy selžou pouze specifickým způsobem popsaným v požadavcích k dependabilitě a pouze do přijatelné míry.

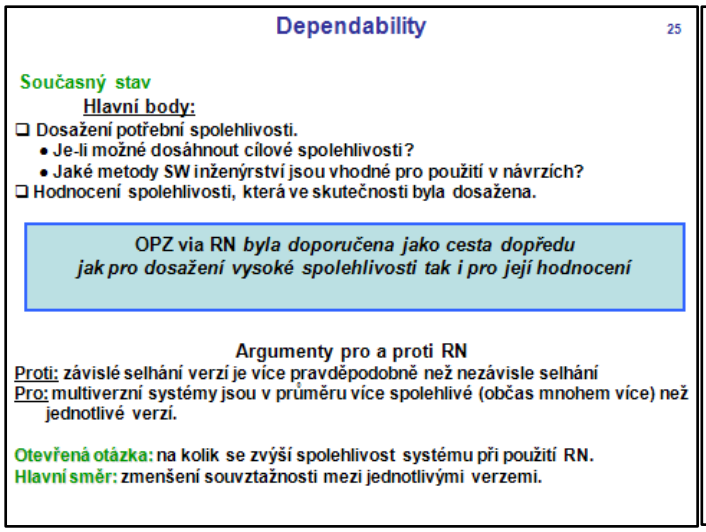
Příklady:
- Stálá výstupní hodnota (nesprávná) na rozdíl od nahodilé výstupní hodnoty;
- absence výstupní hodnoty (ticho) na rozdíl od „blábolení";
- konsistentní na rozdíl od nekonsistentního selhání.

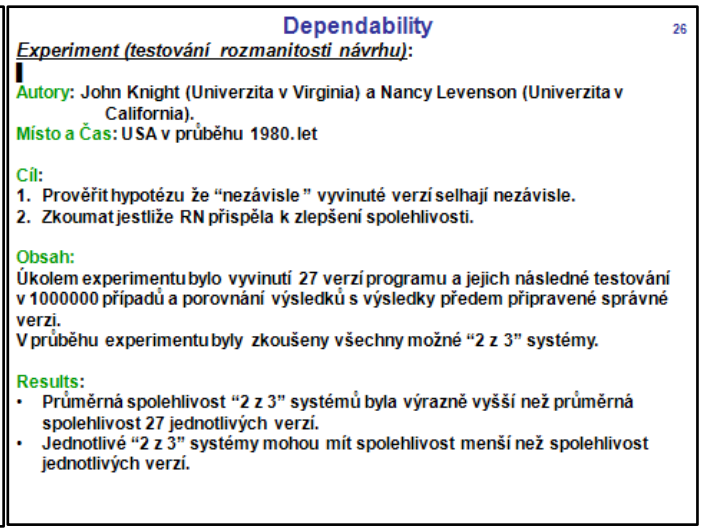
■ Systém jehož selhání se vždycky do přijatelné míry jeví jako zastavení, se jmenuje systém s selhání typu:
"selhání→zastavení" (nebo "selhání→mlčení") Fail-halt (or Fail-silent)

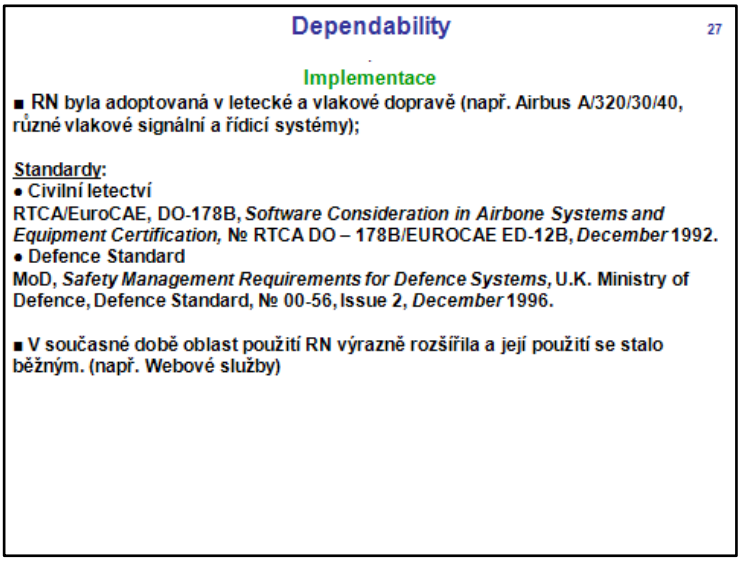
■ Systém jehož selhání jsou do přijatelné míry méně závažná, se jmenuje systém s neškodnými selháními. Fail-safe

Příklady systémů odolných proti závadám

		Odolnost proti závadám
Systémy s vysokou dostupností	Stratus [Wilson 1985]	–
	Tandem SeverNet [Baker et al. 1995]	unikající závady návrhu SW
	IBM S/390 cluster [Brown Associates 1998]	–
	Sun cluster [Bowen et al. 1997]	–
Systémy kritické k bezpečnosti	SACEM Subway speed control [Hennebert & Guiho 1993]	závady návrhu HW a SW a závady kompilátoru
	ELEKTRA Railway signalling system [Kantz & Koza 1995]	závady návrhu HW a SW
	Airbus Flight Control System [Brière & Traverse 1993]	závady návrhu HW a SW
	Boeing 777 Flight Control System [Yeh 1998]	závady návrhu HW a závady kompilátoru







Kontrolní otázky:

- co znamená odolnost systému proti závadám
- vysvětlete hlavní fáze zajištění odolnosti systému proti závadám
- charakterizujte proces obnovení systému

Úkoly pro samostatnou práci:

1) Prostudovat koncepce odolnosti systému proti závadám. Použit literaturu [1], [2] a obsah výkladové části.
2) Vytvořit tabulku s příklady odolnosti systému proti závadám. Pro každý příklad popsat odhalení chyb a obnovení systému (viz Tabulka na slajdu č. 13 ve výkladové části). Tabulka má mít 5 řádků (jeden řádek pro každý příklad) a 3 sloupce. První sloupec pro popis systému, druhý pro popis odhalení chyb, třetí sloupec pro popis obnovení systému.

Kapitola 3. Způsoby zajištění odolnosti informačních systémů proti závadám

Cíl kapitoly:

- vysvětlit hlavní způsoby zajištění odolnosti informačních systémů proti závadám
- probrat různé způsoby zajištění odolnosti informačních systémů na konkrétních příkladech

Klíčová slova: zajištění odolnosti proti závadám, obnovení, maskování, rozmanitost návrhu, opakování

Výkladová část:

Následující prezentace pomůže studentům splnit úkoly samostatné práce.

Odolnost SW proti závadám (OPZ):

1

SW (aplikace), který byl vyvinut, může obsahovat reziduální závady návrhu. Nicméně kvůli dodatečným opatřením tyto závady nebudou vést k selhání aplikace.

Zajištění OPZ:
Všeobecným způsobem zajištění odolnosti SW proti závadám je použití rozmanitosti návrhu, který je založen na používání extra rozmanitého kódu určeného pro odhalení chyb a obnovení aplikace.
Rozmanitost návrhu je takový přístup k vývoji SW, ve kterém dvě a více variant návrhu jsou vyvinuté nezávisle a vyhovují společné specifikaci služby.
Varianty mají za cíl poskytovat stejnou službu ale jsou implementované různými způsoby (za předpokladu že varianty neobsahují stejné závady).
Adjudicator (rozhodčí algoritmus) stanoví (určí) jeden výsledek předpokládaný za správný na základě výsledků různých variant.

Schémata pro zajištění OPZ:

- Obnovovací Bloky (OBy);
- N – variantní programování (NVP);
- N – samokontrolní programování (NSKP).

Obnovovací bloky

2

- Množina variant aplikačního kódu (většinou jeho části) je vyvinutá nezávislými programátory.

Přijímací test (PT) kontroluje správnost vykonávání (výpočtu) varianty.

Toto je zásadně dynamické schéma: varianty mohou být vykonávané výhradně sekvenčně. Tj. jestli výsledek varianty projde přijímacím testem, znamená to, že obnovovací blok je splněný. Jinak se systém odroluje a další varianta (tzv. alternativní varianta) bude zkoušena.

NVP

NVP je statické schéma : všechny varianty se vykonávají souběžně (paralelně) a jejich výsledky se srovnávají Adjudikatorem.

Adjudikator umožňuje najít správnou variantu a maskovat výsledky vadných variant pomocí schématu většinového hlasování.

NSKP

- N samokontrolní SW komponenty jsou vykonané paralelně.

Každý komponent sestává z dvojice variant a porovnávače. Jeden komponent je považován za aktivní komponent, a ostatní za "zatíženou" zálohu.

Obnovovací bloky

3

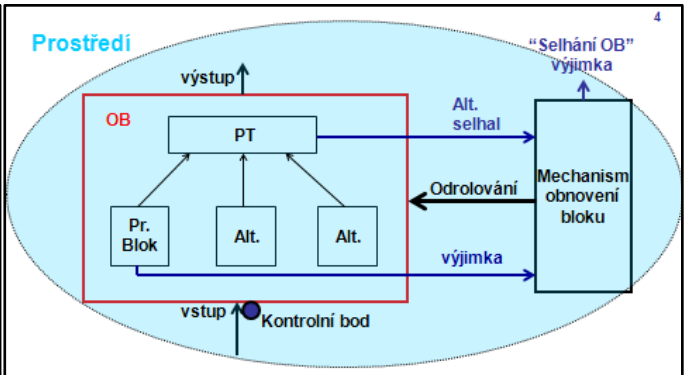
- OB představuje dobře strukturovaný obsah pro operace obnovení po výskytu chyby.

Dobře strukturovaný program sestává z identifikovatelných operací. Většina z nich sama může mít další menší operace.

Cíl:
- zabránit tomu, aby reziduální závady návrhu měly vliv (zapůsobily) na systémové prostředí;
- explicitně uvedení kontrolních operací a také operací, které musí být vykonávany (provedeny) v případě výskytu chyb, a jejich zahrnutí do jediného bloku.

Každý OB sestává z:
- prvořadý blok (nebo primární alternativní blok)
- přijímací test (PT)
- alternativní (náhradní) blok (počet bloků může být 0,1,2,..)

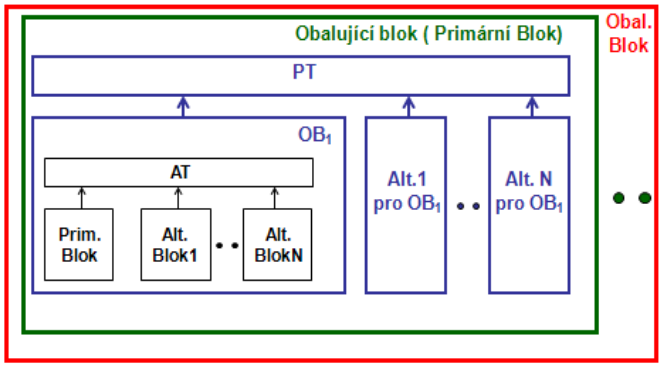
4



- Jestliže všechny alternativní bloky neuspěly u testu nebo v průběhu jejich vykonávání vznikly výjimky jako důsledek odhalení interních chyb, vznikne výjimka "Selhání OB" (failure exception), která bude signalizována v prostředí obnovovacího bloku. Jelikož OBy mohou být vnořené, výjimka „Selhání OB" signalizovaná z vnitřního bloku vyvolí obnovení v obalujícím obnovovacím bloku.

Vložené OBy

5



- Požadavky

OB může být napsán v kterémkoli programovacím jazyce, s použitím libovolného stylu programování nebo metodologie. Jediné požadavky jsou:
- OBy musí být explicitně definované
- OBy musí být dynamicky vnořené
- vstup a výstup bloku musí být explicitně označen

Konverzace

8

Cíl: Vykonávat obnovovací a komunikační operaci koordinovaným způsobem.
- Schéma Konverzace zajišťuje koordinaci spolupůsobících procesů, abychom se vyhnuli „Domino" efektu.

Konverzace probíhá následně:
1) Při vstupu do konverzace každý proces vytváří kontrolní bod;
2) Když chyba bude odhalena v kterémkoli procesu, všechny procesy se vrátí ke svým kontrolním bodům;
3) Po navrácení všechny procesy používají alternativní bloky;
4) Všechny procesy opouští konverzaci společně.

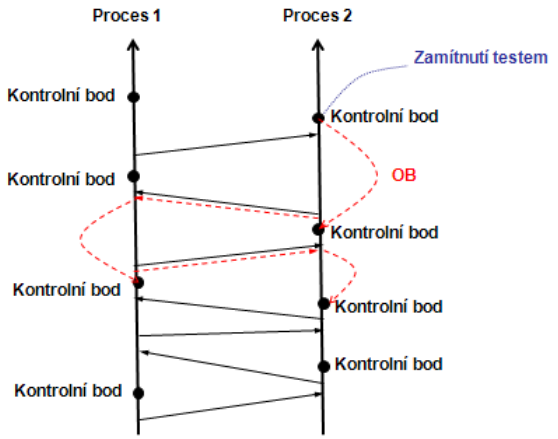
Procesy, které jsou v konverzaci, zůstanou stejné po odhalení chyb a návratu na začátek konverzace. Tytéž procesy po návratu budou používat alternativní moduly.

- Dialog je způsob jak obalit a zahrnout množinu procesů do jedné atomické akce.

- Diskuse (nebo konference) obsahuje množinu dialogů. Diskuse řídí vykonání dialogů a rozhoduje jaké obnovení musí proběhnout, když dialog selže.

Diskuse umožňuje použití různých množin procesů což zajišťuje opravdovou rozmanitost návrhu.

"Domino efekt"

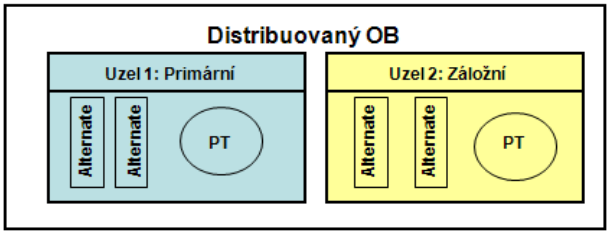


Schémata zajišťující odolnost SW proti závadám

- ✓ Distribuované obnovovací bloky
- ✓ Konsensní obnovovací bloky
- ✓ Opakující bloky s rozmanitými daty
- ✓ Samokonfigurující optimální programování
- ✓ Certifikační stopy
- ✓ t/(n-1) – variantní programování

Distribuované obnovovací bloky

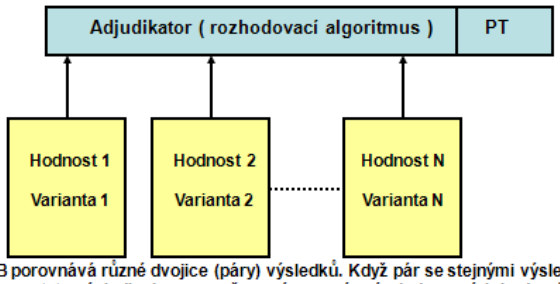
Struktura distribuovaného OB: celý OB je úplně replikován v primárním a záložním (backup) HW uzlech. OB zahrnuje dva alternativní bloky a PT.



Primární uzel používá první alternativní modul jako prvořadý blok, zatímco Záložní uzel používá druhý alternativní modul jako prvořadý blok.

Konsensní OB

- KOB potřebuje mít k dispozici N variant algoritmu. Varianty jsou zhodnocené a vyjmenované (stejně jako v OB) podle pořadí vykonávání a spolehlivosti.



KOB porovnává různé dvojice (páry) výsledků. Když pár se stejnými výsledky bude nalezen, tyto výsledky jsou považované za správné a jeden z nich bude výsledkem KOB. Jestliže žádný pár se stejnými výsledky není nalezen, bude výsledek varianty s nejvyšší hodností zkontrolován přijímacím testem. V případě, že výsledek neprojde testem, další varianta bude vybraná (podle hodnosti), a její výsledek bude zkontrolován přijímacím testem. Tato procedura bude pokračovat pokud všechny varianty nebudou vybrané nebo výsledek jedné z variant projde testem.

Opakující bloky s rozmanitými daty

- Rozmanitost dat je strategie, podle které návrh (algoritmus) se nemění. Algoritmus bude opakován v případě chyb. Při opakování algoritmu budou použitá jiná odlišná data.
- Opakující block vykonává jediný algoritmus a kontroluje jeho výsledek pomocí přijímacího testu.
  - Při úspěchu u testu, opakující blok bude ukončen.
  - Při neúspěchu, algoritmus bude opakován poté, co data budou vyjádřená v jiném formátu.
- Tato procedura bude pokračovat pokud nebudou překročené časové limity nebo výsledek algoritmu projde testem.

Samokonfigurující optimální programování

SOP uspořádá vykonávání SW variant ve fázích, dynamické konfiguruje aktuálně aktivní množiny variant, což umožňuje vytvořit přijatelný (přípustný) výsledek použitím relativně malých prostředků (t.j. pomocí efektivního použití prostředků, které jsou k dispozici).

Certifikační stopy

- Schéma má za cíl vykonat algoritmus tak, aby algoritmus zanechal za sebou stopy dat (certifikační stopy). Tyto stopy (data) budou použité jiným algoritmem, který řeší stejný problém rychleji.
- Výsledky obou algoritmů se porovnávají. Při shodě výsledků první algoritmus je považován za správný.

t/(n-1) – variantní programování

t/(n-1) – diagnostika

Hlavní cíl koncepce: v systému z *n* modulů izolovat **závadné moduly** uvnitř množiny s velikosti nanejvýš **(n-1)**, když počet závadných modulů je nanejvýš **"t"**.

Definice 1. Systém S je ***t-diagnostikovaný*** (na základě vytvořeného syndromu) jestli všechny závadné moduly mohou být identifikované, pod podmínkou že počet závadných modulů nepřekračuje "t".

Definice 2. Systém S je ***t/(n-1)-diagnostikovaný*** (na základě vytvořeného syndromu) jestli všechny závadné moduly mohou být izolované uvnitř množiny s velikosti nanejvýš (n-1) modulů, pod podmínkou že počet závadných modulů nepřekračuje "t".

Adjudikatorní mechanismus

Požadavky k adjudicatoru:
1) Adjudikator a varianty musí být nezávislé, aby se vyloučila možnost společných nebo souvztažných závad;
2) Adjudikator musí být jednoduchý, aby se snížila možnost reziduálních závad.

OB (Přijímací test)	PT je specifický pro každý systém. Je obtížné zajistit aby PT a varianty byli nezávislé navzájem.
NVP (většinové hlasování)	Velmi složitý. Zahrnuje velký počet porovnávání výsledků
NSKP (porovnávač)	Nemůže odhalit souvztažné závady, které se mohou vyskytnout v aktivních samokontrolních komponentech.

t/(n-1) – variantní programování

- Využívá několik nových výsledků z oblasti diagnostiky na systémové úrovni pro návrh jednoduchého adjudikatoru.

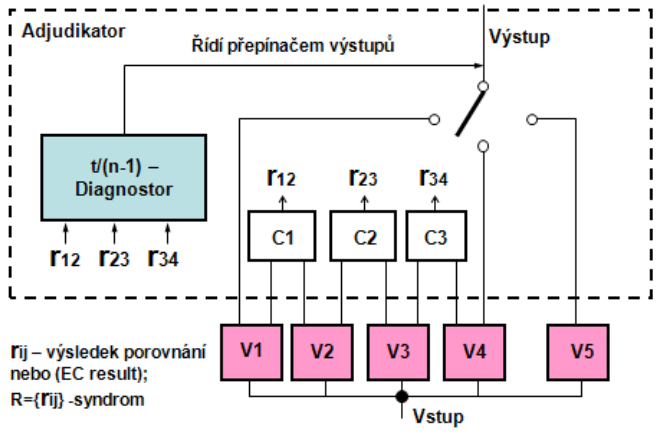
Schéma má několik předností, včetně:
1) Potenciální schopnost odolat více souvztažným závadám mezi variantami;
2) Jednoduchý adjudikator, který vyžaduje pouze O(n) porovnávací kroky;
3) Případně může poskytovat správnou službu dokonce i když počet závadných variant překračuje hranice "t";
4) Dovoluje postupné degradace.

- Obecná t/(n-1)-VP architektura je schopna identifikovat (rozpoznat) správný výsledek v podmnožině výsledků n variant (n SW modulů), pod podmínkou, že počet závadných modulů v architektuře nepřekračuje "t".

Sémantika t/(n-1) – VP:

- Všechny n nezávislé navržené SW varianty se vykonávají paralelně.
- Jen některé z výsledků variant se porovnávají aby se vytvořil syndrom.
- Na základě syndromu diagnostický program vykonává t/(n-1)-diagnostiku (t.j. určí správnou variantu) a vybere výsledek varianty, která je pokládaná za správnou, pomocí přepínání výsledků. Jestliže správná varianta nebude identifikovaná, bude použitá náhradní varianta (pokud existuje), nebo bude signalizovaná výjimka.

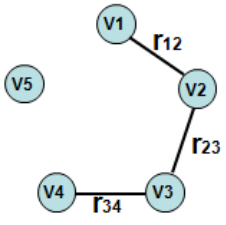
Příklad: n=5 a t=2 (2/(5-1) – VP)



Všechny možné syndromy a výběr výsledků pro 2/(5-1)-VP architekturu

r_{12}	r_{23}	r_{34}	Výsledky předpokládané za správné
0	0	0	r_1, r_2, r_3, r_4
0	0	1	r_1, r_2, r_3
0	1	0	r_5
0	1	1	r_1, r_2
1	0	0	r_2, r_3, r_4
1	0	1	r_5
1	1	0	r_3, r_4
1	1	1	r_5

- Alespoň jeden z výsledků r1, r4 a r5 je předpokládaný za správný pro každý syndrom!

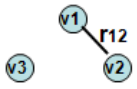


V1, V2, V3, V4, V5 – Varianty

C1, C2, C3 – Porovnávače

$r_{i,i+1}$ – výsledek porovnání

r_{ij} = 0 (1) jestli výsledky variant Vi a Vj jsou totožné (nejsou totožné)



V případě n=3 a t=1 stačí pouze jeden porovnávač pro t/(n-1)-diagnostiku.

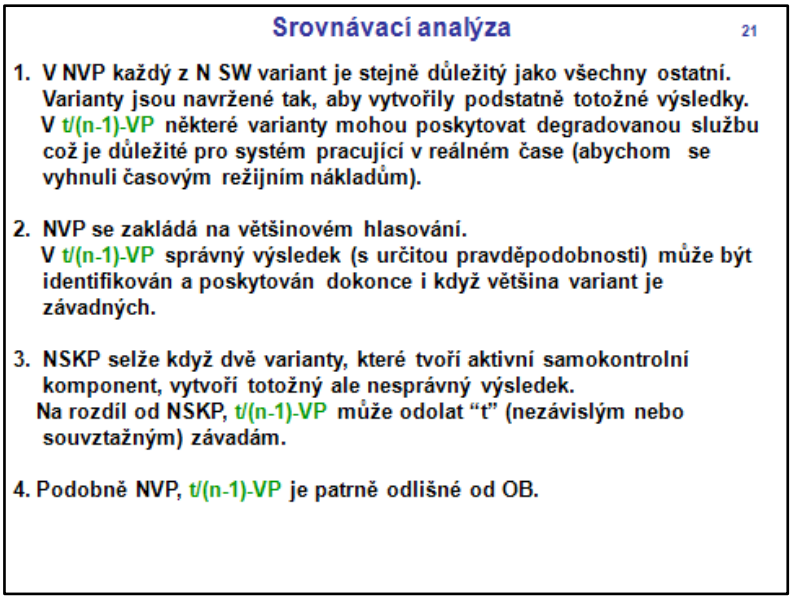
- Porovnávání výsledků může být organizované ve formě řetězů, kde porovnávač Ci (1≤ I ≤ n-1) porovnává výsledky variant V_i and V_{i+1}

Další možnost: představení struktury porovnání výsledků ve formě grafu $H_{2r,n}$, kde výsledek varianty Vi (1≤ I ≤n) se porovnává s výsledkem varianty V_j jenom když if i-r ≤ j ≤ i+r (mod n+1, r = 1,2,3,..).

- Smyčka je $H_{2,n}$ graf (r = 1).

Věta: Systém **S** složený z **n** modulů (SW variant) je t/(n-1) – diagnostikovaný jestli $n \geq 2t + 1$ a struktura porovnávání výsledků obsahuje přinejmenším (alespoň):
1. Řetěz z t + 1 modulů pro t=1;
2. Řetěz z t + 2 modulů pro t=2;
3. Řetěz z 2t + 1 modulů pro 3 ≤ t < 5;
4. $H_{2r,n}$ strukturu s r=1 pro 5 ≤ t < 7 (t.j. smyčka);
5. $H_{2r,n}$ strukturu s $r \geq (t-1)/5$ pro 7≤ t.

Pro reálné hodnoty n (3≤ n ≤10), t/(n-1) – VP architektura používá pouze O(n) porovnávačů a jednoduchý algoritmus s lineární složitostí.



Kontrolní otázky:
- charakterizujte základní schémata zajišťující odolnost systému proti závadám
- pojmenujte a popište další schémata (sekundární) pro zajištění odolnosti systému proti závadám
- proveďte srovnávací analýzu schémat

Úkoly pro samostatnou práci:
Prostudovat způsoby zajištění odolnosti informačních systémů proti závadám. Použit literaturu [1] a obsah výkladové části.

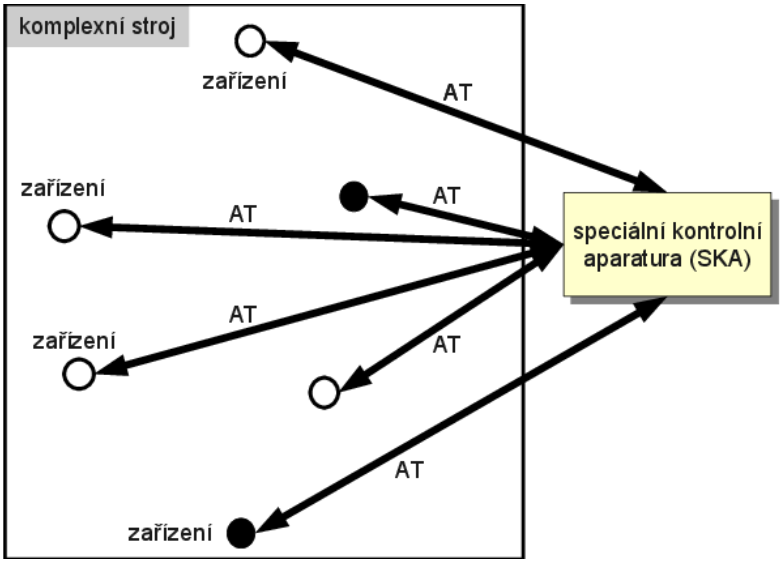
Kapitola 4. Samokontrola a samodiagnostika na systémové úrovni

Cíl kapitoly:
- vysvětlit samokontrolu a samodiagnostiky na systémové úrovni
- procvičit návrh samodiagnostiky systémů na konkrétních příkladech

Klíčová slova: samokontrola, samodiagnostika, metody, algoritmy a organizace samokontroly a samodignostiky

Podkapitola 4.1. Podstata a základní prvky samokontroly a samodiagnostiky

Výkladová část:
Současný svět je plný složitých technických programovatelných zařízení, které řídí neméně komplexní stroje jako jsou letadla, vlaky a v poslední době i osobní automobily. Tato technická zařízení tvoří hardware a programové vybavení, pro něž je klíčová bezchybná činnost. Důležitou roli proto hraje **kontrola** těchto zařízení.
Technická diagnostika uvažuje dvě fáze kontroly zařízení: před průběhem vlastní činnosti zařízení (*předběžná*) a v jeho průběhu (*průběžná*).
Předběžná kontrola může být prováděna pomocí *speciální kontrolní aparatur*y (SKA), která detailně prověří technický stav daného zařízení (viz Obr. 4.1) je typické především pro předstartovní diagnostiku letadel (testovací aparatura je v tomto případě rozsáhlé externí zařízení, jehož instalace přímo v letounu by byla neefektivní).



Obrázek 4.1. Speciální kontrolní aparatura

Samotná předběžná kontrola může zahrnovat vícenásobnou výměnu dat mezi SKA a zařízením a jejich následné zpracování kontrolním algoritmem v SKA. Tento kontrolní algoritmus obvykle provádí jednoduché porovnání výstupních dat ze zařízení s daty, která jsou považována za správná (etalonní).
Tato celková a mnohdy komplexní kontrola modulu, zahrnující jak výměnu dat tak provádění kontrolního algoritmu, může být na vyšší úrovni abstrakce interpretována jako jediná **atomická akce resp. kontrola** (AT = *atomic test* nebo *elementary check*). Na vyšší, tj. systémové úrovni abstrakce, jsou detaily kontroly irelevantní, v kontextu této abstrakce tudíž vystačíme s pohledem, že jeden systémový modul kontroluje druhý.
SKA je obvykle považována za bezchybnou, tj. můžeme plně důvěřovat výsledkům kontroly. V praxi je toho dosahováno různými způsoby, přičemž jedním z nejdůležitějších kritérií je úplnost kontroly. Například při kontrole programu s mnoha větvemi toku řízení (např. pro různá vstupní data) je nutno projít všemi větvemi. Pokud by byla některá větev vynechána, nebude kontrola úplná.
Hlavní výhodou SKA je proto vysoká důvěryhodnost výsledků. Naopak k nevýhodám předběžné kontroly za použití SKA patří:

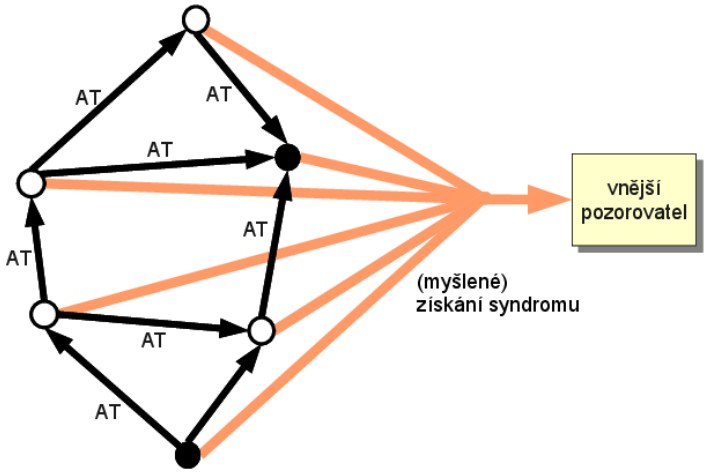
- vysoké (finanční) náklady a náročné použití
- časové a režijní náklady
- nutnost zaškolení personálu (kontrola běžně vyžaduje účast operátora)
- velké prostorové nároky běžných kontrolních mechanismů.

Nyní se zaměříme na zajímavější druhou fázi, to jest **průběžnou kontrolu**. Kontrola je v tomto případě prováděna souběžně s hlavní činností zařízení. V tomto případě nelze běžně použít speciální kontrolní aparaturu (projevují se všechny výše uvedené nevýhody), její náhrada je však komplikovaná.
Jedním z řešení je použití specializovaného modulu umístěného externě, tj. vně kontrolovaného zařízení. Oproti SKA je specializovaný modul výrazně jednodušší a jeho funkce jsou omezené. Tento modul provádí kontrolu a sběr dat ve vymezených časových intervalech.
Hlavní nevýhodou použití kontrolního modulu je obtížné zajištění vysoké spolehlivosti a důvěryhodnosti výsledků kontroly. Navíc v průběhu provádění hlavní činnosti kontrolovaná zařízení navzájem komunikují, což ovlivňuje a ztěžuje jeho kontrolu.
Možným řešením výše uvedených problémů je **samodiagnostika,** tj. vzájemná kontrola a diagnostika jednotlivých účelových zařízení komplexního stroje. V tomto případě neexistuje žádné dedikované kontrolní zařízení, tj. všechna zařízení vykonávají jak běžnou tak kontrolní činnost.
Atomická kontrola může i v tomto případě zahrnovat:

1. jednoduchou kontrolu přijatých běžných dat (např. kontrolní součty)
2. složitější kontrolu přijatých dat (např. testování etalonu)
3. odesílání speciálních kontrolních dat a přijetí a zpracování odezvy.

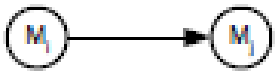
Na systémové úrovni abstrakce odpovídá každému zařízení **modul**. Celkový model systému je tak representován grafem, v němž uzly representují moduly tj. jednotlivá dílčí zařízení a hrany atomické kontroly. Tento graf se nazývá **diagnostickým grafem** systému (zkráceně **DG)**
Každé technické zařízení (tj. modul) může být buď ve stavu, kdy poskytuje správná výstupní data (= **bezchybný modul**) resp. kdy jsou jím produkovaná data nesprávná (= modul selhal, **chybný modu**l). Bohužel může kontrolující zařízení chybně vyhodnotit data přijatá ze zařízení kontrolovaného a to v obou směrech (správná označit za chybná resp, chybná za správná) a tak zařízení nesprávně ohodnotit.

Každý modul tak má svá vlastní hodnocení modelů, které zkontroloval a tato hodnocení nemusí být konzistentní (tj. nemusí existovat konsenzus v hodnocení jednotlivých modulů). Navíc pravděpodobnost nekonzistence může být relativně velká.
Necháme prozatím stranou problémem nekonzistence v hodnoceních jednotlivých modulů, ale zamyslíme se jak je možno i přes případnou nekonzistentnost využít výsledků jednotlivých kontrol.
Na začátku si pro jednoduchost představíme, že existuje **abstraktní vnější pozorovatel**, který dostane výsledky všech dílčích atomických kontrol (viz obrázek 4.2). Předpokládejme pro jednoduchost, že přitom nedojde k žádnému chybnému přenosu nebo chybné interpretaci obdržených dat. Tento pozorovatel ve skutečnosti neexistuje, neboť kontrola musí být plně autonomní. Jeho zavedení však zjednoduší počáteční model systému.



Obrázek 4.2. Vnější pozorovatel

Lze si tak například položit otázku, zda je tento abstraktní pozorovatel schopen na základě výsledků atomických kontrol určit, které moduly jsou bezchybné a které naopak selhaly. Tento problém je již součástí systémové diagnostiky.
Diagnostika na **systémové úrovni** spočívá v odhalení všech chybných (=selhávajících) modulů. Naopak je nutno zdůraznit, že na této úrovni se neuvažuje konkrétní příčina selhání modulu (tj. co se stalo uvnitř zařízení).
V uvedeném zjednodušeném modelu je jediným vstupem diagnostiky (prováděné abstraktním vnějším pozorovatelem) množina výsledků atomických kontrol. Tato množina je označována jako **syndrom**. Výstupem je seznam chybných modulů, resp. komplementární seznam modulů chybných.
Výstup, tj. výsledek diagnostiky i jeho důvěryhodnost, závisí na několika předpokladech souvisejících s atomickými kontrolami. Nejdůležitějším je předpoklad o výsledcích kontrol prováděných jak bezchybnými tak selhávajícími moduly.
Uvažujme například atomickou kontrolu modulu M_j provedenou modulem M_i (viz obrázek 4.3). Předpokládejme, že výsledek kontroly bezchybného modulu bezchybným modulem je roven vždy hodnotě **0**. Obvyklá notace má tvar $r_{ij} = 0$.



Obrázek 4.3. Atomická kontrola

Složitější je situace v případě kontroly selhávajícího modulu (zde tedy např. *Mj*) modulem bezchybným (zde např. *Mi*). Většinou se předpokládá, že r_{ij} je v tomto případě rovno **1**, tj. bezchybný modul vždy odhalí chybu v modulu selhávajícím [4].
Nakonec uvážíme situaci, kdy kontrolu provádí selhávající modul. V tomto případě lze předpokládat, že výsledek bude náhodný, tj. může nabývat jak hodnoty 0 tak 1. V nejjednodušším případě budou tyto hodnoty nabývat se stejnou pravděpodobností. Existují však i další modely, například Barsi, Grandoni a Masstrini [5], nabízejí v tomto případě předpoklad, že výsledek této kontroly je vždy roven hodnotě 1 (tj. kontrolovaný modul bude vždy označen za chybný). V praxi navíc můžeme mít i zpřesňující informace o chování selhávajícího modelu, včetně pravděpodobnosti produkování výsledků atomických kontrol (tj. např. zpřesněnou informaci o produkování zavádějících výsledků).

Všechny výše uvedené předpoklady navíc uvažují, že spojení mezi moduly jsou bezchybná (tj. při přenosu nedochází k ztrátě nebo modifikaci informací). V opačném případě by musely být předpoklady přehodnoceny. Zde však budeme vycházet pouze z následujících relativně jednoduchých předpokladů (podle Preparata).

DEFINICE 1.1:

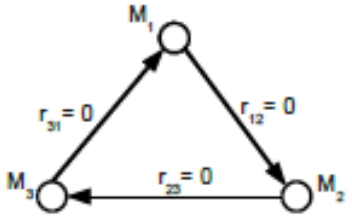
Výsledek atomické kontroly modulu *Mj* modulem *Mi* je definován takto:

$$r_{ij} = \begin{cases} 0 & \text{jsou-li oba moduly } M_i \text{ i } M_j \text{ bezchybné} \\ 1 & \text{je-li } M_i \text{ bezchybný a } M_j \text{ selhávající} \\ X(0,1) & \text{je-li } M_i \text{ selhávající} \end{cases}$$

Výsledek, který poskytuje vnˇejší pozorovatel, je ovlivnˇen i strukturou atomických kontrol, které tvoˇrí *diagnostický graf.*

DEFINICE 1.2:

Syndrom *R* je uspořádaná množina výsledků jednotlivých atomických kontrol tj. $R=\{r_{ij}\}$. Jednotlivé výsledky r_{ij} se označují jako prvky syndromu. Jednotlivé prvky syndromu mohou být v diagnostickém grafu representovány jako ohodnocení hran, kde se hodnota rovná výsledku atomické kontroly. Viz obr. 4.4, kde ohodnocení pro přehlednost obsahuje i označení výsledků.



Obrázek 4.4. Diagnostický graf systému

Tento diagnostický graf presentuje syndrom v názorné formě. Usnadňuje provedení diagnostické analýzy (z pozice vnějšího pozorovatele. Například syndrom na obrázku 4.4 umožňuje učinit závěr, že všechny tři moduly jsou bezchybné. Samozřejmě jen tehdy, pokud platí zvolený model ohodnocení atomických kontrol.

<u>*Kontrolní otázky*</u>:

- co je atomická kontrola
- definujte diagnostický graf systému
- charakterizujte výsledek atomické kontroly

<u>*Úkoly pro samostatnou práci*</u>:

1) Prostudovat pojmy: atomická kontrola, diagnostický graf systému. Použit literaturu [1] a obsah výkladové části.
2) Vytvořit v programovacím jazyce C# objektový model reprezentující systém se samodiagnostikou. Jádrem by měly být instance třídy *Module*, implementující reprezentaci chybových stavů a metod pro atomické kontroly a objekt třídy *System*, reprezentující množinu modulů včetně diagnostického grafu. Inspiraci můžete najít v opoře „Datové struktury a algoritmy samokontroly v Pythonu“, kapitola 2.

Podkapitola 4.2. Problém určení množiny atomických kontrol. Hodnocení diagnostických grafů

<u>*Výkladová část*</u>:

Pokud je k dispozici diagnostický graf, je možno položit si otázku, zda lze systém diagnostikovat tj. určit chybné moduly, a to bez ohledu na obdržený syndrom. Lze dokázat, že úspěšnost diagnostiky závisí na počtu chybných tj. selhávajících modulů. Tento počet se označuje jako *t*. Pro některé hodnoty *t* lze vždy vytvořit diagnostický graf, který bude s úplnou jistotou zaručovat správnou diagnostiku bez ohledu na získaný syndrom.

Pro grafy, u nichž je zaručeno diagnostikování *t* chybných modulůu, tzv. **t-diagnostikovatelnost**, platí následující *nutná* podmínka:

V t-diagnostikovatelném grafu musí být modul kontrolován nejméně *t* dalšími moduly, přičemž v grafu nejsou vícenásobné hrany (= atomické kontroly).

Pokud však počet chybných modulů *t* překročí jisté t_{max} , pak již není možné takový DG zkonstruovat. Preparata ve své práci [4] dokázal, že pro toto t_{max} platí:

$$t_{max} = \lfloor (N-1)/2 \rfloor$$

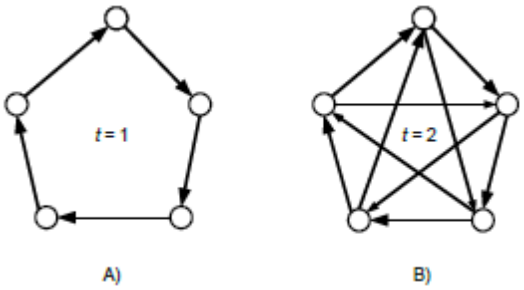
kde N je počet uzlů resp. modulů.
t-diagnostikovatelných grafů (pro $t \cdot t_{max}$) však může existovat i více. Zajímavé jsou však především ty s malým či dokonce nejmenším počtem atomických kontrol (větší počet kontrol prodražuje a komplikuje diagnostiku)
DEFINICE 1.3:
Diagnostický graf **t-optimální**, pokud obsahuje minimální počet hran, které stačí pro zajištění určité hodnoty t . Pro hodnotu $t = t_{max}$ je možno graf stručně označit jako **optimální**.
Počet hran *t-optimálního grafu* lze snadno vypočítat podle následujícího vztahu:

$$l = tN$$

jenž lze v případě *optimálního diagnostického grafu* dále upravit na:

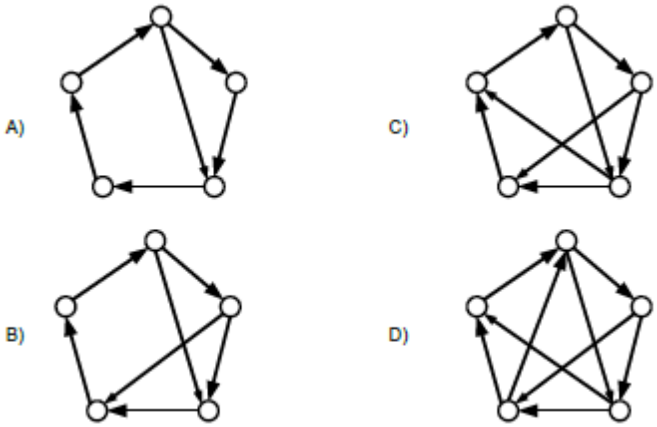
$$l = t_{max} N = \lfloor (N-1)/2 \rfloor N$$

kde
N = počet uzlů (modulů);
t_{max} = maximální hodnota parametru *t;*
l = počet hran optimálního DG.
Libovolný graf, který obsahuje více než l hran, je podle této definice považován za nadbytečný. Vztah lze použít i v opačném směru a pro danou hodnotu parametru t vytvářet instance struktur DG, které zajišťují určité diagnostické vlastnosti, například schopnost odhalit určitý počet chybných modulů. Například diagnostický graf na obr. 4.5(A) je t-optimální pro $t = 1$ neboť zajišťuje detekci pouze jednoho chybného modulu a počet hran je pro dané t minimální, DG na obrázku 4.5(B) má $t = 2$ a zajišťuje tak již detekci dvou chybných modulů (a je navíc optimální neboť $t = t_{max}$ a počet hran je minimální)



Obrázek 4.5.: Optimální diagnostické grafy

Všechny diagnostické grafy zobrazené na obr. 4.6 zajišťují detekci stejného počtu chybných modulů ($t = 1$), ale mají různý počet hran. Metoda hodnocení DG navržená Preparatou neumožňuje porovnat tyto diagnostické struktury a definovat přesněji jejich diagnostické vlastnosti. Zohledňuje se pouze dosažená hodnota t resp. t-optimálnost.



Obrázek 4.6.: Diagnostické grafy *s* t=1

Kromě parametru t však existují také další *kritéria* pro hodnocení diagnostických vlastností grafu, které umožňují porovnání a ohodnocení grafů na obrázku 4.6. Například každému DG může byt přiřazena hodnota pravděpodobnosti, která bude odrážet diagnostické schopnosti grafu. Konkrétně je to pravděpodobnost, že syndrom odpovídající určitému DG umožní správně diagnostikovat stav všech modulů. Tuto pravděpodobnost si vysvětlíme pomocí jednoduchého příkladu.

Nechť má například DG strukturu zobrazenou na obrázku 4.7. Z obrázku je zřejmé, že když bude bezchybný jen modul M_1, pak obdržený syndrom umožňuje správně diagnostikovat stavy ostatních modulů (tj, že jsou oba chybné). Pravděpodobnost této situace můžeme spočítat pomocí obecného vztahu:

$$P(A_K) = C_K^N \cdot (1-P_M)^K \cdot P_M^{N-K} \cdot \frac{C_K}{C_K^N} = (1-P_M)^K \cdot P_M^{N-K} \cdot C_K$$

kde

P_M = pravděpodobnost, že modul je v chybném stavu (selže). Pravděpodobnost je u všech modulů stejná.

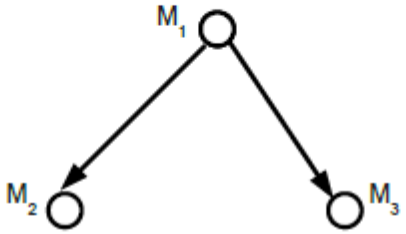
K = počet bezchybných modulů (zde 1)

C_K = počet možností výběru podgrafu tvořeného K uzly, ze kterého jsou všechny ostatní uzly přímo dosažitelné

C_K^N = počet kombinací k-té třídy z n prvků $\binom{n}{k}$

V případě jednoprvkové množiny uzlů ($K = 1$) je pravděpodobnost rovna:

$$P(A_1) = (1-P_M) \cdot P_M^2 \cdot C_1$$



Obrázek 4.7.: Ukázkový DG pro vysvětlení pravděpodobnosti P_{SD}

Pro DG na obrázku 4.7 se číslo C_1 rovná 1, protože existuje pouze jeden výběr jednoprvkové množiny uzlů, z nichž jsou ostatní uzly přímo dosažitelné (množina $\{M1\}$).
Správnou diagnostiku získáme také v případě, že jsou správné pouze dva moduly ze tří a to buď $\{M_1,M_2\}$ nebo $\{M_1,M_3\}$. Pravděpodobnost této události je rovna:

$$P(A_2) = (1-P_M)^2 \cdot P_M \cdot C_2$$

Pro uvažovaný diagnostický graf je C_2 rovno 2, neboť existují pouze dva podgrafy s dvěma uzly, z nichž jsou dosažitelné všechny ostatní uzly ($\{M_1,M_2\}$, $\{M_1,M_3\}$). To však ještě není vše, neboť správný výsledek přirozeně dostaneme i v případě, že jsou správné všechny tři moduly. Pravděpodobnost takovéto události je:

$$P(A_2) = (1-P_M)^3 \cdot P_M^0 \cdot C_3 = (1-P_M)^3$$

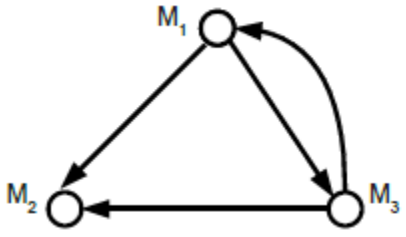
C_3 je v tomto případě vždy rovno 1, a to bez ohledu na strukturu atomických kontrol. Výsledek diagnostiky systému je správný, pokud nastane *alespoň jedna* ze situací A_K , $k = 1, \ldots, n$. Z toho vyplývá, že pravděpodobnost P_{SD} získání správného výsledku diagnostiky systému na základě syndromu, který odpovídá DG je rovna:

$$P_{SD} = \sum_{K=1}^{N} P(A_K) = \sum_{K=1}^{N} (1-P_M)^K \cdot P_M^{N-K} \cdot C_K$$

Nyní již můžeme vypočítat pravděpodobnost P_{SD} pro diagnostický graf na obr. 4.7.
Například pro $P_M = 0.1$:

$$P_{SD} = (1-P_M) \cdot P_M^2 \cdot 1 + (1-P_M)^2 \cdot P_M \cdot 2 + (1-P_M)^3 = 0.9$$

Nyní uvážíme nový mírně rozšířený diagnostický graf z obrázku 4.8.

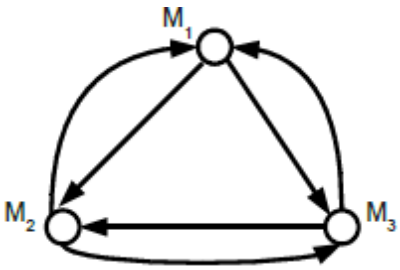


Obrázek 4.8.: Rozšířený ukázkový DG (přidány dvě hrany)

Zde byly přidány dvě hrany. Modul M_3 nyní kontroluje ostatní moduly (M_1a M_2). Pro tento DG se proto změní čísla C_1 a C_2:
C_1 = 2 výběry : {M_1, M_2}
C_2 = 3 výběry: {M_1,M_2}a {M_1,M_3} a {M_2,M_3}
Pravděpodobnost P_{SD} pro tento DG a shodnou hodnotu P_M = 0.1 je rovna (1- P_M).

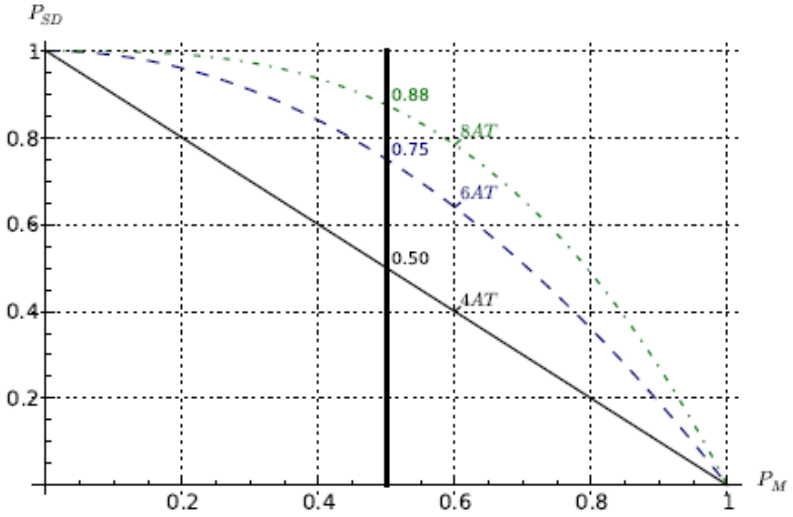
$$P_M^2 \cdot 2 + (1 - P_M)^2 \cdot P_M \cdot 3 + (1 - P_M)^3 = 0.99.$$

Nakonec uvážíme diagnostický graf z obrázku 4.9. Zde byly přidány další dvě atomické kontroly z modulu M_2. Číslo C_1 se tak zvýší na 3 (z každého uzlu lze přímo dosáhnout ostatní), Číslo C_2 zůstává na hodnotě 3. Pravděpodobnost P_{SD} se opět mírně zvýší a to na 0.993.



Obrázek 4.9.: Rozšířený ukázkový DG (přidány čtyři hrany)

Přidání hran zde tedy nevede ke zvýšení *t*, neboť to je rovno t_{max} . Zvýší se však pravděpodobnost získání správného výsledku diagnostiky. To lze ještě lépe vidět z grafu závislosti P_{SD} na P_M na obrázku 4.10. S rostoucí chybovostí modulů P_M pravděpodobnost správné diagnostiky systému P_{SD} klesá, ale u DG s větším počtem atomických testů je pokles méně výrazný, Jednotlivé křivky odpovídají DG na obrázcích 4.7 (dole), 4.8 (uprostřed) a 4.9 (nahoře).
Při výpočtu pravděpodobnosti správného výsledku diagnostiky P_{SD} hrají klíčovou roli čísla C_K , která odrážejí strukturu diagnostického grafu a tudíž i strukturu atomických kontrol. Tato čísla se označují jako č*ísla charakteristická*.
DEFINICE 1.4:
Charakteristické číslo C_K, k = 1, 2, . . . ,n je *počet výběru K uzlů* (podgrafů) z diagnostického grafu, ze kterých jsou všechny *ostatní uzly přímo dosažitelné*.
U jednoduchých grafů lze charakteristická čísla zjistit snadno z nákresu diagnostického grafu. U komplexnějších diagnostických graf je však nutno využít automatizovaného výpočtu nad modifikovanou maticí sousednosti. *Modifikovaná matice sousednosti* je odvozena z běžné matice sousednosti nastavením hodnoty 1 u všech prvků na diagonále (tj. je zohledněn fakt, že uzel je dostupný ze sebe sama).



Obrázek 4.10.: Funkční závislost P_{SD} na P_M a počtu AT

Algoritmus musí otestovat všechny kombinace výběrů, tj. jde o vyčerpávající prohledání, a vypočítat charakteristické číslo podle následujícího vztahu:

$$C_K = \frac{1}{k!} \underbrace{\sum_{i_k=1}^{n} \sum_{i_{k-1}=1}^{n} \cdots \sum_{i_1=1}^{n}}_{i_k \neq i_{k-1} \neq \cdots \neq i_1} \left[\prod_{j=1}^{n} (a_{i_k j} \vee a_{i_{k-1} j} \vee \ldots \vee a_{i_1 j}) \right]$$

Použití daného vztahu si ukážeme na modifikované matici sousednosti grafu z obrázku 4.8.
Při výpočtu charakteristického čísla *Ci* se vypočítává suma přes všechny řádky matice, při čemž pro každý řádek je počítán produkt přes sloupce.

$$C_i = \sum_{i=1}^{3} \left[\prod_{j=1}^{3} a_{ij} \right], \text{kde j=číslo sloupce}$$

$a_{11}=1$	$a_{12}=1$	$a_{13}=1$
$a_{21}=0$	$a_{22}=1$	$a_{23}=1$
$a_{31}=1$	$a_{32}=0$	$a_{33}=1$

Hodnota produktu pro první řádek je rovna 1, neboť $a_{11} \cdot a_{12} \cdot a_{13} = 1$. Produkt u ostatních řádků je roven 0 ($a_{21} \cdot a_{22} \cdot a_{23} = 0$, $a_{31} \cdot a_{32} \cdot a_{33} = 0$). Suma pro C_1 je tedy rovna 1.
V případě čísla C_2 je nutné otestovat všechny uspořádané dvojice řádků tj. variace bez opakování (1,2), (1,3), (2,1), (2,3), (3,1), (3,2).

$$C_2 = \frac{1}{2!} \sum_{i_2=1}^{3} \sum_{i_1=1}^{3} \left[\prod_{j=1}^{3} (a_{i_2 j} \vee a_{i_1 j}) \right]$$

Proměnné i_2 a i_1 určují jednotlivé dvojice řádků (prochází se všechny kombinace dvojic). Poté se v jednotlivých sloupcích spočítá logický součet hodnot, a to vždy jen v řádcích i_1 a i_2, a z výsledných hodnot je vypočítán produkt. Pro první dvojici (1,2) je například potřeba spočítat následující produkt:

$$\prod_{j=1}^{3} (a_{2j} \vee a_{1j}) = 1$$

Podobně se vypočítají i další dvojice.
Program pro výpočet charakteristického čísla *Cn* pro konkrétní *n* je relativně snadný (má *n* +2 cyklů). Obecný výpočet je složitější, neboť vyžaduje generování velkého kombinací. Následující ukázka je v jazyce Haskell:

```
1  import Data.List (delete, transpose)
2
3  permutations :: Eq a => Int -> [a] -> [[a]]  -- permutations
4  permutations 0 _  = [[]]
5  permutations n xs = [x:xs | x<-xs, xs <- permutations (n - 1) (x `delete` xs)]
6
7  ladd :: Int -> Int -> Int -- logical or
8     -- (logical OR i.e. || is defined only for boolean values)
9  ladd 0 0 = 0
10 ladd _ _ = 1 --for all other = 1
11
12 getRows :: [Int] -> [a] -> [a] --get row with indices
13 getRows indices matrix = [ row | (i, row) <- (zip [0..] matrix),
14                                  i `elem` indices]
15
16 cnum :: Int -> [[Int]] -> Int --characteristic number
17 cnum n matrix =
18   (sum [ product ( map (foldr1 ladd) (transpose (getRows i matrix)))
19            | i <- permutations n [0..v]]) `div` f
20        where v = length matrix - 1
21              f = product [1..n] --factorial
```

Program definuje tři pomocné funkce. Funkce *combinations* produkuje všechny n-prvkové kombinace v podobě seznamu seznamů. Funkce *ladd* definuje logický součet nad celými čísly (resp. nad podmnožinou {0,1}). Funkce *getRow* umožňuje výběr řádků z matice (representované jako seznam seznamů). Vyjímané řádky jsou dány seznamem indexů (parametr *indices*).
Vlastní výpočet charakteristického čísla (funkce *cnum*) je již relativně přímočará. Je zde počítána suma seznamu, který je získán aplikováním dílčího výrazu na všechny *n*-prvkové kombinace řádku. Je použita tzv. seznamová komprehenze, v níž je na proměnnou *i* , která postupně odkazuje na jednotlivé kombinace aplikován výraz před svislítkem. V rámci výrazu se nejdříve provede výběr řádku podle aktuální kombinace index°u (*getRows*) a výsledek jenž je opět maticí je transponován.
Původní sloupce se tak stávají řádky a tak může být proveden výpočet logických součinů všech hodnot v řádcích pomocí mapování funkcionálu *foldr1* na jednotlivé (souvislé) řádky. *Foldr1* aplikuje binární funkci, která je uvedena jako první parametr (zde logický součet *ladd*), mezi všemi hodnotami daného řádku (tj. provede výpočet

$$a_{i_k j} \vee a_{i_{k-1} j} \vee \ldots \vee a_{i_1 j}$$

pro dané *j*). Následně je vypočítán produkt ze seznamu výsledků této funkce pro všechny řádky (= původní sloupce). Vyčerpávající prohledávání všech n-tic je pomalé a neefektivní, neboť výpočetní složitost je exponenciální. Částečně jej lze urychlit využitím již vypočítaných dílčích hodnot. Mnohé hodnoty jako např. řádkové součiny resp. dílčí sloupcové počty jsou v průběhu výpočtu využívány vícenásobně. Ani toto urychlení však nemusí být dostatečné v situacích, kdy je graf rozsáhlý a doba výpočtu je limitována, neboť výsledek musí být k dispozici "okamžitě". Proto se stále hledají nové metody výpočtu charakteristických čísel. Pro tyto ú˘cely lze využít různé invariantní charakteristiky diagnostického grafu např. spektrum grafu.

Kontrolní otázky:

- co je optimální diagnostický graf
- co znamená t-diagnostikovatelnost
- co to je charakteristické číslo pro diagnostický graf

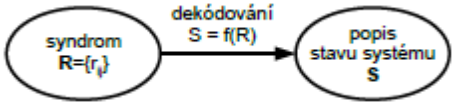
Úkoly pro samostatnou práci:

1) Prostudovat hodnocení diagnostických grafů. Použit literaturu [1] a obsah výkladové části.
2) Doplnit objektovou representaci systému se samodiagnostikou (viz úkoly v podkapitole 4.1) o metody pro hodnocení grafu (t-diagnostikovatelnost, t-optimálnost, o pravděpodobnost, že syndrom odpovídající určitému DG umožní správně diagnostikovat stav všech modulů). Při implementaci můžete vycházet z implementace v opoře „Datové struktury a algoritmy samokontroly v Pythonu“, kapitola 4 (jádrem řešení pro výpočet charakteristických čísel je iterátor přes všechny k-prvkové variace, který není na rozdíl od Pythonu v C# obsažen ve standardní knihovně)

Podkapitola 4.3. Návrh diagnostických algoritmů

Výkladová část:

Provedení atomických kontrol a získání syndromu není konečným cílem. Hlavním cílem je zjištění stavu systému, tj. nalezení všech chybných resp. správných modulů. Po provedení atomických kontrol a získání syndromu nastupuje další fáze, vlastní diagnostika systému. Tato fáze je nezbytná, pokud je výsledkem jedné nebo více atomických kontrol hodnota „1“, neboť to svědčí o přítomnosti chybných modulů v systému.
V uvažovaném modelu se předpokládá, že diagnostiku bude provádět myšlený externí pozorovatel, který získává syndrom ze systému. Externí pozorovatel není součástí systému, je sám bezchybný a bezchybný je i přenos údajů ze systému k vnějšímu pozorovateli.
Cílem samodiagnostiky je zjištění, jaký modul, resp. jaké moduly selhaly, respektive zpřesnění informace o druhu chyby. V některých případech není možné určit konkrétní chybné moduly, ale je možné pouze stanovit podmnožinu modulů, v níž jsou chybné moduly soustředěny, nicméně však může obsahovat i moduly bezchybné.
Libovolnou diagnostiku tak lze popsat jako funkci převádějící získaný syndrom na popis stavu systému $S = f(R)$, respektive jako dekódování syndromu na popis stavu (viz obrázek 4.11).
Toto dekódování je dále závislé na různých předběžných informacích o stavu systému, včetně apriorních předpokladů o chování systému v různých stavech, o vlastnostech jednotlivých atomických kontrol, nebo o režimech selhání modulů systému apod.



Obrázek 4.11. Podstata diagnostiky

Algoritmy samodiagnostiky, které se používají v praxi, zohledňují především následující dodatečné informace:

- předpoklad o výsledcích atomických kontrol prováděných jak správnými tak i selhávajícími moduly
- pořadí provádění množiny atomických kontrol
- spolehlivost jednotlivých modulů systému a spolehlivost spojení mezi moduly
- předpoklad o maximálním počtu chybných modulů. Tento předpoklad určuje mez, za kterou mohou být výsledky samodiagnostiky s určitou pravděpodobností považovány za chybné. Respektive lze na základě

požadované jistoty určit mez, v jejímž rozsahu lze s danou pravděpodobností předpokládat, že zjištěný stav systému odpovídá skutečnosti.
- předpoklad o režimech selhání jednotlivých modulů systému. Většinou je uvažováno, zda jsou selhání jednotlivých modulů řízená či nikoliv. Selhání modulů mohou být například stálá, přechodná nebo nahodilá.
- možností obnovení systému (tj. nahrazení chybného modulu nebo jeho průběžné odstranění).

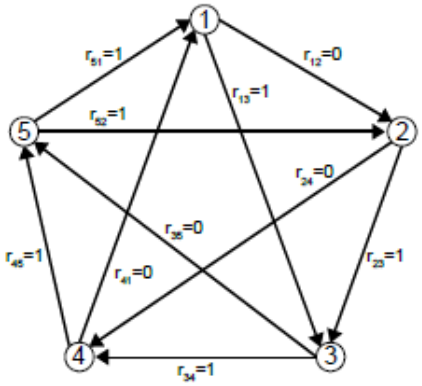
První obecné algoritmy samodiagnostiky začaly být navrhovány počínaje rokem 1967, kdy byl publikován Preparatův článek [4]. Od té doby byla navržena celá řada dalších algoritmů samodiagnostiky. Následující přehled se věnuje těm nejzákladnějším a nejužitečnějším.

Algoritmy založené na tabulce syndromu

Tabulkové algoritmy pracují s **tabulkou syndromu**, což je maticová representace syndromu. Tabulka syndromu $M_R[r_{ij}]$ je čtvercová matice o rozměru $N \times N$, kde N je počet modulů. Pokud je součástí syndromu výsledek atomické kontroly, kterou provádí i-tý modul na modulu j-tém, to jest hodnota r_{ij}, pak tabulka syndromu obsahuje tuto hodnotu v i-tém řádku a j-tém sloupci. Položka v tomto případě obsahuje hodnotu nula nebo jedna.

Pokud není atomická kontrola mezi určitými dvěma moduly systému provedena (tj. není v syndromu k dispozici), pak je tato situace graficky representována pomlčkou v průsečíku příslušného sloupce a řádku. Při representaci tabulky syndromu v počítači lze použít například hodnotu -1.

Obrázek 4.12 znázorňuje dvě možné representace syndromu. Vlevo representaci pomocí ohodnoceného diagnostického grafu, vpravo pomocí ekvivalentní tabulky (matice) syndromu.



$M_R =$

	1	2	3	4	5	
	-	0	1	-	-	1
	-	-	1	0	-	2
	-	-	-	1	0	3
	0	-	-	-	1	4
	1	1	-	-	-	5

Obrázek 4.12.: Dvě varianty representace syndromu

Pokud je graf t-diagnostikovatelný, lze pomocí tabulkových algoritmů identifikovat všechny chybné moduly, ale samozřejmě pouze v případě, že jejich počet nepřekročí hodnotu t. V opačném případě bude algoritmus s velkou pravděpodobností schopen tuto situaci detekovat, ale chybné moduly nemohou být identifikovány (tj. program může nanejvýše signalizovat, že počet chybných modulů je příliš velký). Výsledkem však může být i zcela zmatečná identifikace chybných modulů.

Přípravným krokem tabulkových algoritmů je proto určení hodnoty t z diagnostického grafu. Běžnější je však využití ad hoc navrženého diagnostického grafu se zaručenou hodnotou t, která je obvykle zároveň optimální, tj. je rovno $t_{max} = \lfloor (N-1)/2 \rfloor$.

Před volbou a použitím tabulkového algoritmu je navíc nutné zohlednit vlastnosti atomických kontrol. Většina tabulkových algoritmů pracuje s vlastnostmi AT podle definice Preparata [4] .

Při volbě konkrétního tabulkového algoritmu je rozhodující počet modulů v systému, neboť tyto algoritmy jsou na počtu modulů silně závislé. Větší počet modulů může algoritmus výrazně zkomplikovat, a tak může být čas provedení diagnostiky pro větší počet modulů neakceptovatelný (např. s exponenciální časovou složitostí), resp. výrazně závislý na konkrétním syndromu. Další vývoj se proto zaměřuje na návrh tabulkových algoritmů s akceptovatelnou a predikovatelnou časovou složitostí.

Algoritmy založené na tabulce potenciálních syndromů

Dalším příkladem tabulkových algoritmů jsou algoritmy založené na *tabulce potenciálních syndromů*. Jeden z prvních algoritmů tohoto typu byl navržen Vedeshenkovem [6].

Tabulka potenciálních syndromů se vytváří před začátkem diagnostické procedury. Podkladem pro vytvoření tabulky je matice sousednosti diagnostického grafu a reprezentace vlastností atomické kontroly (většinou se volí opět reprezentace podle definice Preparata).

Tabulka potenciálních syndromů zahrnuje všechny syndromy, které mohou být obdrženy pro různé přípustné kombinace stavů modulů. Zpravidla se opět předpokládá, že počet chybných modulů v systému nepřekročí hodnotu t. V tomto případě stačí uvažovat jen ty kombinace stavů, v nichž je počet chybných modulů menší než t. Počet stavů a tudíž i potenciálních syndromů je v tomto případě roven:

$$Q = \sum_{i=1}^{t} \binom{n}{i}$$

Tak například pro systém, jehož diagnostický graf je zobrazen na Obr. 4.12, budou uvažovány pouze následující situace:

S_1: M_1 je chybný
S_2: M_2 je chybný
S_3: M_3 je chybný
S_4: M_4 je chybný
S_5: M_5 je chybný
S_6: M_1 a M_2 jsou chybné
S_7: M_1 a M_3 jsou chybné
S_8: M_1 a M_4 jsou chybné
S_9: M_1 a M_5 jsou chybné
S_{10}: M_2 a M_3 jsou chybné
S_{11}: M_2 a M_4 jsou chybné
S_{12}: M_2 a M_5 jsou chybné
S_{13}: M_3 a M_4 jsou chybné
S_{14}: M_3 a M_5 jsou chybné
S_{15}: M_4 a M_5 jsou chybné

Když skutečný stav systému neodpovídá žádné z uvažovaných situací (například v případě, když je počet chybných modulů větší než dva) je výsledek diagnostiky nesprávný nebo dokonce zavádějící. Diagnostické algoritmy, které používají tabulky potenciálních syndromů, proto mají jen omezenou důvěryhodnost. Tuto důvěryhodnost však lze předem spočítat.

Tabulka potenciálních syndromů má následující sloupce:

- označení uvažované situace (S_i)
- čísla chybný modulů $\{M_j\}, j = 1 \ldots n$ pro danou situace S_i
- označení potenciálního syndromu (R^i_p) pro danou situaci S_i
- jednotlivý prvky syndromu $\{r_{ij}\}$ pro danou situaci, tyto prvky tvoří l sloupců, kde l je počet atomických kontrol (resp. hran).

Tabulka potenciálních syndromů obsahuje tolik řádků, kolik je uvažovaných situací. Hodnoty jednotlivých prvků potenciálního syndromu $\{r_{ij}\}$ jsou stanoveny podle zvolené reprezentace atomické kontroly. V případě Preparatovy representace mohou jednotlivé prvky nabývat hodnot 0,1 nebo X v závislosti na stavech modulů M_i a M_j. Hodnota označovaná jako X vyjadřuje náhodný výsledek kontroly prováděné chybným modulem. Ve skutečném syndromu může nabývat hodnoty 0 nebo 1 s určitým náhodným rozdělením. Pravděpodobnostní charakteristiky atomických kontrol nejsou pro tento algoritmus podstatné.

Pro systém s diagnostickým grafem zobrazeným na obrázku 4.12 je tabulka potenciálních syndromů následující:

S_i	chybné moduly	potenciál. syndrom	atomické kontroly									
			r_{12}	r_{13}	r_{23}	r_{24}	r_{34}	r_{35}	r_{45}	r_{41}	r_{51}	r_{52}
S_1	M_1	r_p^1	x	x	0	0	0	0	0	1	1	0
S_2	M_2	r_p^2	1	0	x	x	0	0	0	0	0	1
S_3	M_3	r_p^3	0	1	1	0	x	x	0	0	0	0
S_4	M_4	r_p^4	0	0	0	1	1	0	x	x	0	0
S_5	M_5	r_p^5	0	0	0	0	0	1	1	0	x	x
S_6	M_1, M_2	r_p^6	x	x	x	x	0	0	0	1	1	1
S_7	M_1, M_3	r_p^7	x	x	1	0	x	x	0	1	1	0
S_8	M_1, M_4	r_p^8	x	x	0	1	1	0	x	x	1	0
S_9	M_1, M_5	r_p^9	x	x	0	0	0	1	1	1	x	x
S_{10}	M_2, M_3	r_p^{10}	1	1	x	x	x	x	0	0	0	1
S_{11}	M_2, M_4	r_p^{11}	1	0	x	x	1	0	x	x	0	1
S_{12}	M_2, M_5	r_p^{12}	1	0	x	x	0	1	1	0	x	x
S_{13}	M_3, M_4	r_p^{13}	0	1	1	1	x	x	x	x	0	0
S_{14}	M_3, M_5	r_p^{14}	0	1	1	0	x	x	1	0	x	x
S_{15}	M_4, M_5	r_p^{15}	0	0	0	1	1	1	x	x	x	x

Jak lze z tabulky snadno vidět, kterékoli dva potenciální syndromy (tj. kterékoli dva řádky v tabulce) jsou odlišné alespoň v jednom prvku. Tato odlišnost syndromů je klíčová pro diagnostiku na základě potenciálních syndromů. Vlastní algoritmus je prováděn po skončení běhu atomických kontrol, to jest po získání skutečného syndromu. Algoritmus spočívá v porovnání skutečného syndromu se syndromy potenciálními. Cílem je nalézt potenciální syndrom, který odpovídá syndromu reálnému, což umožní identifikovat chybné moduly (jsou uvedeny v druhém sloupci tabulky). Při porovnání se musí shodovat všechny výsledky atomických kontrol, nejednoznačný výsledek u

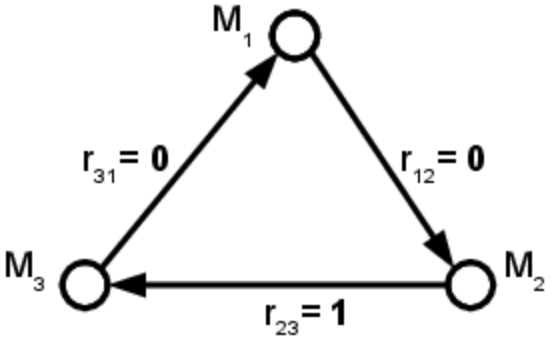
potenciálního syndromu (označený jako „x“) se shoduje s libovolným výsledkem reálné kontroly (žolíkové porovnávání).
Pro porovnávání existuje několik strategií. Základní a nejjednodušší strategie, jež spočívá v postupném porovnávání reálného syndromu s jednotlivými řádky tabulky, je neefektivní, neboť vyžaduje největší počet porovnání (maximálně až $Q \cdot l$).
V našem ukázkovém případě, v němž je reálný syndrom roven $R_A = \{r_{12}=0, r_{13}=1, r_{23}=1, r_{24}=0, r_{34}=1, r_{35}=0, ,r_{45}=1, ,r_{41}=0, ,r_{51}=1, r_{52}=1\}$, lze i při použití základní strategie snadno nalézt shodující se potenciální syndrom r_p^{14} a tím diagnostikovat stav modulů v systému (chybné jsou moduly M_3, M_5, správné M_1, M_2, M_4).
I při ručním prohledávání tabulky se však jako výhodnější jeví postupný předvýběr řádků pomocí několika prvních hodnot syndromu (například, pokud zohledníme jen první prvek syndromu, omezí se výběr na 11 potenciálních řádků).
Tento algoritmus předvýběrů lze snadno rozšířit a implementovat. Nejdříve jsou vybrány řádky, u nichž se shoduje první prvek s reálným syndromem (jsou to řádky 1, 3, 4, 5, 6, 7, 8, 9, 13, 14, 15). V druhém kroku se zaměříme jen na vybrané řádky a testujeme shodu u druhého prvku syndromu. Výběr se opět omezí na řádky (1, 3, 6, 7, 8, 9, 13, 14). Postupný výběr pokračuje a končí v sedmém kroku, v němž se rozhodne mezi variantami potenciálních syndromů r_p^3 a r_p^{14}. Počet porovnání je v tomto případě výrazně menší [B].
Na závěr této sekce shrneme některé přednosti a nevýhody tabulkových algoritmů plynoucí z jejich návrhu i použití:
výhody:

- potřebují pouze základní informace o systému (ty máme zpravidla vždy k dispozici)
- tabulky mohou být snadno vytvořeny a jsou názorné, což snižuje chybovost při jejich zpracování

nevýhody:

- tabulkové algoritmy nezohledňují spolehlivost jednotlivých modulů systému, což snižuje důvěryhodnost výsledku

Pravděpodobnostní algoritmy
Pravděpodobnostní algoritmy samodiagnostiky jsou zaměřeny na výpočet aposteriorní pravděpodobností stavů jednotlivých modulů systému. Po zjištění daných pravděpodobností může být učiněno rozhodnutí buď o stavech všech modulů v systému, nebo alespoň o některých modulech (v případě nedostatku informací). Důvěryhodnost výsledků může být navíc zvýšena zohledněním předběžných informacích o spolehlivosti jednotlivých modulů. Z tohoto důvodu nejsou pravděpodobnostní algoritmy omezeny jen na t-diagnostikovatelné grafy s maximálně t chybnými moduly, ale mohou poskytovat relevantní informace i při nižším počtu atomických kontrol, nebo při větším počtu chybných modulů.
K návrhu pravděpodobnostních algoritmů přispěli především H. Fujiwara a K. Kinoshita, kteří již v roce 1981 navrhli jednoduchý a efektivní algoritmus [7].
Nejdříve si připomeňme, že pro zjištění stavu modulu můžeme použít jak *apriorní* tak *aposteriorní* pravděpodobnost. *Apriori* znamená doslova „před“. Proto přívlastek apriorní vyjadřuje pravděpodobnost určitého stavu modulu ještě před provedením atomických kontrol. Tato pravděpodobnost je ve většině případů stanovena na základě dodatečných informací, například informace o spolehlivosti modulu. Tato informace bývá uváděna v dokumentaci k modulu. Běžně to bývá např. parametr λ, tj. intenzita exponenciálního rozdělení selhání. Apriorní pravděpodobnosti se mohou u jednotlivých modulů lišit. Tím se pravděpodobnostní přístup k diagnostice liší od přístupu tabulkového, který vychází z předpokladu, že všechny moduly mají stejnou (apriorní) pravděpodobnost selhání. Zahrnutím specifického chování jednotlivých modulů se může zvýšit důvěryhodnost diagnostiky, neboť ta již například nemusí být omezena hodnotou t.
Aposteriorní pravděpodobnost naproti tomu vyjadřuje pravděpodobnost určitého stavu modulu po provedení jeho kontroly. Je zřejmé, že aposteriorní pravděpodobnost správného stavu u správného modulu je vyšší než apriorní, neboť výsledky kontroly přidávají další informace o stavu modulů, čímž zvyšují naši jistotu o stavu systému.
Základem algoritmu je výpočet aposteriorní pravděpodobnosti jednotlivých stavů (správný/chybný) u všech modulů v systému. Pro tento účel využijeme jednoduchý příklad diagnostického grafu s třemi moduly (M_1, M_2, M_3) a třemi atomickými kontrolami. Diagnostický graf je znázorněn na obrázku 4.13.



Obrázek 4.13.: Ukázkový DG a syndrom pro popis pravděpodobnostního algoritmu

Výpočet aposteriorních pravděpodobností stavů modulů začíná výpočtem aposteriorní pravděpodobností hypotéz všech možných stavů modulů. Zde je pouze poněkud zjednodušen model apriorních pravděpodobností a tím zkrácena symbolika.

Předpokládejme, že apriorní pravděpodobnosti bezchybného stavu modulů jsou známé a nabývají hodnoty P_1,P_2 a P_3. Symboly q_1, q_2 a q_3 označují apriorní pravděpodobnost, že moduly jsou v chybném stavu. Je zřejmé, že $q_1 = 1 - P_1$, $q_2 = 1 - P_2$ a $q_3 = 1 - P_3$. Dále předpokládejme syndrom podle obrázku 4.13.

1. určení všech možných hypotéz ohledně stavu modulů. V našem případě se všechny moduly mohou nacházet jak ve stavu správném tak chybném. Je tak nutno uvažovat osm hypotéz:

H_1: 123 M_1 je správný M_2 je správný M_3 je správný

H_2: 123 M_1 je správný M_2 je správný M_3 je chybný

H_3: 123 M_1 je správný M_2 je chybný M_3 je správný

H_4: 123 M_1 je správný M_2 je chybný M_3 je chybný

H_5: 123 M_1 je chybný M_2 je správný M_3 je správný

H_6: 123 M_1 je chybný M_2 je správný M_3 je chybný

H_7: 123 M_1 je chybný M_2 je chybný M_3 je správný

H_8: 123 M_1 je chybný M_2 je chybný M_3 je chybný

2. výpočet pravděpodobnosti všech hypotéz

$P(H_1) = P_1P_2P_3$
$P(H_2) = P_1P_2q_3$
$P(H_3) = P_1q_2P_3$
$P(H_4) = P_1q_2q_3$
$P(H_5) = q_1P_2P_3$
$P(H_6) = q_1P_2q_3$
$P(H_7) = q_1q_2P_3$
$P(H_8) = q_1q_2q_3$

Protože hypotézy $H_1 \ldots H_8$ popisují všechny možné situace, je jejich celková pravděpodobnost rovna jedné.

3. určení podmíněných pravděpodobností

Výpočet provádíme pro událost, v níž atomické kontroly $\tau_{12}, \tau_{23}, \tau_{31}$ skončí s výsledkem (syndromem) podle obrázku 4.13. Proto musíme nejdříve vypočítat pravděpodobnost získání tohoto syndromu za podmínky, že stavy modulů odpovídají určité hypotéze. Jednotlivé podmíněné pravděpodobnosti i zde závisí na representaci výsledků atomických kontrol. Pokud využijeme klasickou representaci Preparatovu a navíc budeme předpokládat, že pravděpodobnost jedničkového výsledku je vždy $P_r = P\{X = 1\}$ (tj. výsledek kontroly prováděné chybným modulem nezávisí na stavu kontrolovaného modulu), získáme následující pravděpodobnosti:

$P(R/H_1) = 0$ $\quad P(R/H_5) = 0$

$P(R/H_2) = 1 - P_r$ $\quad P(R/H_6) = (1 - P_r)^2$

$P(R/H_3) = 0$ $\quad P(R/H_7) = 0$

$P(R/H_4) = 0$ $\quad P(R/H_1) = (1 - P_r)^2 P_r$

Nulové pravděpodobnosti jsou u hypotetických stavů, které nemohou daný syndrom produkovat. Například, pokud by byly všechny moduly správné (hypotéza H_1), pak by nemohl správný modul M_2 označit za chybný modul M_3 (výsledek atomické kontroly r_{23} je jedna). Hypotéza H_6 je naproti tomu slučitelná se syndromem a pravděpodobnost je součinem tří nezávislých pravděpodobností: $P(R{:}r_{12} = 0/H_6)$, což je pravděpodobnost, že chybný modul M_1 (viz hypotéza) provede kontrolu modulu M_2 s výsledkem „1“ $= (1 - P_r)$; $P(R{:}r_{23} = 1/H_6) = 1$, neboť správný modul vždy odhalí chybný; a nakonec $P(R{:}r_{31} = 0/H_6) = (1 - P_r)$ ze stejných důvodů jako výše (hypotéza předpokládá, že M_3 je chybný). Pravděpodobnosti hypotéz H_2 a H_8 lze vyjádřit obdobným způsobem.

4. určení podmíněné pravděpodobnosti hypotéz při daném syndromu

Pro výpočet jednotlivých podmíněných pravděpodobností $P(H_i/R)$ lze využít Bayesův vztah. Jmenovatel zlomku vyjadřuje pravděpodobnost syndromu R při daných apriorních pravděpodobnostech, tj. $P(R) = \sum_{i=1}^{\ell} P(H_i)P(R/H_i)$. Pro náš příklad je $P(R)$ rovno $(1 - P_r)P_1P_2q_3 + (1 - P_r)^2 q_1P_2q_3 + (1 - P_r)^2 P_r q_1q_2q_3$.

Pravděpodobnosti jednotlivých hypotéz za podmínky získání daného syndromu mají následující tvar (nulové jsou vynechány):

$$P(H_2/R) = \frac{(1-P_r)P_1P_2q_3}{P(R)}, \quad P(H_6/R) = \frac{(1-P_r)^2q_1P_2q_3}{P(R)},$$

$$P(H_8/R) = \frac{(1-P_r)^2P_rq_1q_2q_3}{P(R)}$$

5. určení aposteriorní pravděpodobnosti správnosti jednotlivých modulů

Aposteriorní pravděpodobnost, že je určitý modul správný, lze získat z podmíněných pravděpodobností jednotlivých hypotéz. Tato pravděpodobnost je totiž rovna podmíněné pravděpodobnosti události, v níž stav systému odpovídá libovolné hypotéze, která daný modul považuje za správný. Například v našem případě je modul M_2 považován za správný v hypotézách H_1, H_2, H_5 a H_6 .

Podmíněnou pravděpodobnost sjednocení hypotéz lze získat součtem podmíněných pravděpodobností těchto hypotéz (podmínka je ve všech případech stejná, po provedení atomické kontroly je získán určitý syndrom). Když aposteriorní pravděpodobnost správnosti modulu M_i označíme symbolem P^*_i, pak můžeme napsat:

$$P_i^* = \sum_{H \in U} P(H/R), \text{ kde } U = \{H_j : M \text{ je správné v } H_j\}$$

V našem případě má pro modul M_2 vztah tuto konkrétní podobu:

$$P_2^* = P(H_1/R) + P(H_2/R) + P(H_5/R) + P(H_6/R) = \frac{(1-P_r)P_1P_2q_3 + (1-P_r)^2q_1P_2q_3}{(1-P_r)P_1P_2q_3 + (1-P_r)^2q_1P_2q_3 + (1-P_r)^2P_rq_1q_2q_3}$$

Pro názornost můžeme aposteriorní pravděpodobnost vyčíslit pro konkrétní hodnoty apriorních pravděpodobností například pro $P_1 = P_2 = 0.8$. Dále předpokládejme, že pravděpodobnost P_r je rovna 0.5 (chybný modul vrací při kontrole ostatních modulů se stejnou pravděpodobností buď hodnotu „0“ nebo „1“).

Podle uvedeného vztahu se aposteriorní pravděpodobnost správnosti modulu M_2 rovná 0.986.

Jak lze vidět, po provedení trojice atomických kontrol se zvýšila naše jistota o správnosti modulu z 0.8 (apriorní pravděpodobnost) na téměř 0.99, neboť atomické kontroly byly v souladu s apriorním předpokladem.

Dílčí pravděpodobnosti a výsledky pro všechny moduly ukazuje obrázek 4.14 (získaný z tabulkového kalkulátoru *OpenOffice Calc*).

P_r	0,50
$1-P_r$	0,50

	P_i	q_i	P_i^*
M_1	0,80	0,20	**0,877**
M_2	0,80	0,20	**0,986**
M_3	0,80	0,20	**0,000**

		$P(H_i)$	$P(R/H_i)$	$P(H_i/R)$
H_1	123	0,512	0,000	0,000
H_2	123	0,128	0,500	0,877
H_3	123	0,128	0,000	0,000
H_4	123	0,032	0,000	0,000
H_5	123	0,128	0,000	0,000
H_6	123	0,032	0,250	0,110
H_7	123	0,032	0,000	0,000
H_8	123	0,008	0,125	0,014

$P(R)$	0,073

Obrázek 4.14.: Výpočet aposteriorní pravděpodobnosti správnosti modulů

Výpočet aposteriorních pravděpodobností správnosti jednotlivých modulů je však pouze podkladem vlastní diagnostiky. Hlavním cílem je stejně jako u výše uvedených diagnostických algoritmů identifikace správných i chybných modulů.

Kontrolní otázky:

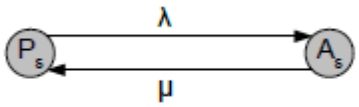
- vysvětlete podstatu diagnostiky na základě množiny výsledků atomických kontrol

- charakterizujte algoritmy založené na tabulce syndromu
- charakterizujte pravděpodobnostní algoritmy

Úkoly pro samostatnou práci:
1) Prostudovat návrh diagnostických algoritmů. Použit literaturu [1] a obsah výkladové části.
2) Implementovat tabulkový a pravděpodobnostní algoritmus v modelu systému se samokontrolou (viz úkoly v podkapitole 4.1). Vzorem mohou být implementace v opoře „Datové struktury a algoritmy samokontroly v Pythonu“, kapitoly 7,8).

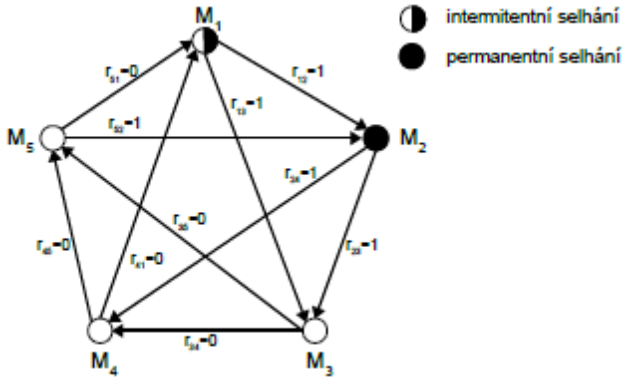
Podkapitola 4.4. Diagnostika intermitentních selhání

Výkladová část:
Intermitentní selhání modulů má na rozdíl od selhání permanentních, tj. trvalých, občasný resp. přerušovaný charakter. Důvody takového chování mohou být různé a představují specifickou oblast výzkumu. Zde pouze poznamenejme, že jednou z příčin může být například vliv radiace.
Pro naše účely jsou mnohem zajímavější matematické modely intermitentně selhávajících modulů. Jeden z takových modelů byl navržen Mallelem a Massonem [8]. Tento model uvažuje dva stavy intermitentního selhání, a to stav pasivní P_s a stav aktivní A_s. Dále používá dva číselné parametry λ a μ určující přechody mezi těmito stavy (viz obr.4.15).



Obrázek 4.15. Model intermitentních selhání

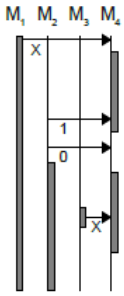
Tento i další modely se používají k modelování a testování metod diagnostiky intermitentních selhání.
Pro diagnostiku intermitentních selhání je možno použít metody používané u stálých selhání (viz výše), je však nutno uvažovat tři stavy modulů: správný, permanentní selhání a intermitentní selhání. U pravděpodobnostního algoritmu je tak např. nutno uvažovat 3^n hypotéz, což může být výpočetně velmi náročné (a to i v případě malých systémů, neboť např. 3^{10} je řádově rovno 10^{10}). Kromě toho může použití pravděpodobnostních algoritmů vést k situaci, kdy budou mít dvě hypotézy o stavu modulu stejnou resp. podobnou aposteriorní pravděpodobnost, což by vyžadovalo další kritéria pro rozhodnutí. Tato situace může vzniknout především v případě stejných apriorních pravděpodobností u jednotlivých modulů.
V případě tabulkových algoritmů je situace ještě složitější, neboť intermitentní chování porušuje základní předpoklad Preparatovy representace: bezpodmínečné odhalení chybného modulu modulem správným. To může vést k nesprávnému nebo dokonce zcela zmatečnému výsledku diagnostiky. Tato situace je ilustrována pomocí obrázku 4.16, jenž popisuje systém s pěti moduly. V systému předpokládáme jeden modul s permanentním selháním a jeden se selháním intermitentním.



Obrázek 4.16. Systém s intermitentními selháními

Získaný syndrom je kompatibilní s výše uvedeným předpokladem o stavu modulů. Pokud však máme k dispozici pouze tento syndrom, nelze učinit rozhodnutí, který z modulů M_1 a M_3 je správný a který má intermitentní selhání. Nalezení obecného intermitentního selhání je velmi obtížné, neboť charakter selhání (vyjádřitelný například pomocí parametrů λ a μ) se může výrazně měnit. Pro některé druhy intermitentních selhání však existují speciální metody, které nalezení a identifikaci chybných modulů výrazně usnadňují.

Navíc je nutno zdůraznit, že v případě intermitentních selhání je důležitá nejen vlastní identifikace selhávajícího modulu, ale i stanovení dalšího postupu pro zacházení s tímto modulem. I zde existují specifické třídy selhání, včetně intermitentní selhání, u nichž je vysoká pravděpodobnost, že modul může být používán i nadále bez jakéhokoliv druhu opravy.
Na základě modelování intermitentních selhání modulů s parametry λ a μ a se zohledněním časových okamžiků atomických kontrol t_{AT} lze intermitentní selhání rozdělit do následujících tří druhů:
1.druh zahrnuje intermitentní selhání, která mohou být odhalena při několika málo (2 až M) opakovaných atomických kontrolách
2.druh intermitentní selhání, která sice mohou být odhalena po opakovaných kontrolách, avšak těchto kontrol však musí být relativně velký počet (až např. v řádu 10^6)
3.druh intermitentní selhání, u nichž je možnost zachycení velmi nepravděpodobná, resp. zachycení nastává nejvýše jedenkrát během diagnostiky.
Souběh intermitentních selhání a atomických kontrol lze nejlépe znázornit na modifikovaném sekvenčním diagramu, jenž je znám například z modelovacího jazyka UML. Ukázkový diagram je na obrázku 4.17.



Obrázek 4.17. Sekvenční diagram intermitentního systému

Jednotlivé moduly jsou v tomto diagramu zobrazeny jako svislé čáry. Ve směru ze shora dolů roste čas. Atomické kontroly jsou znázorněny jako vodorovné čáry se šipkami. Jejich vertikální poloha (osa y) určuje čas provedení atomické kontroly, přičemž atomická kontrola ležící výše je provedena před atomickou kontrolu ležící níže. Počátek a konec šipky (v horizontálním směru) určuje modul provádějící kontrolu (počátek) a modul kontrolovaný (konec). Pod šipkou je zapsán obdržený syndrom (syndromy na obrázku odpovídají definice Preparata [4], kde X je náhodná veličina).
Pro příklad, první (nejčasnější) atomická kontrola je prováděna modulem M_1 a kontrolován je modul M_4. Výsledkem může být jednička nebo nula.
Selhání je v grafu znázorněno tmavě šedým obdélníkem u svislice daného modulu, jenž pokrývá časové období, v němž modul selhává. Modul M_1 je permanentně chybný po celou dobu diagnostiky systému, u modulu M_2 se také jedná o selhání permanentní, jenž vzniká až v průběhu diagnostiky systému a diagnostiku nijak neovlivňuje, neboť modul se po selhání již neúčastní atomických kontrol. Modul M_3 selže jen na velmi krátký okamžik a pouze jednou. Jedná se tedy o intermitentní selhání třetího druhu (je zachyceno jedenkrát). U modul M_4 se opět jedná o intermitentní selhání, které se však opakuje a výrazněji ovlivňuje diagnostiku. Díky malému počtu atomických kontrol nelze stanovit, zda se jedná o selhání prvého nebo druhého druhu.
Pro diagnostiku intermitentních selhání prvého druhu byly navrženy metody založené na sumárním syndromu R_Σ, jenž může být získán po M-násobně opakovaném provedení množiny atomických kontrol.
Sumární syndrom spočítáme takto:

$$R_\Sigma = \{r^*_{ij}\},\ r^*_{ij} = \bigvee_l r^l_{ij},\ \text{kde}\ r^l_{ij} \in R_l (\text{syndrom obdržený při } l\text{-tém opakování})$$

Operace „∨“ je zobecněný logický součet. Jinak řečeno, pokud bude v jednom opakování AT zjištěn u atomické kontroly výsledek „0“ a ve druhém opakování „1“, je sumární výsledek roven 0∨1 = 1 a modul je označen za chybný.
Diagnostika může být provedena pouze v případě, že bude splněna následující podmínka:

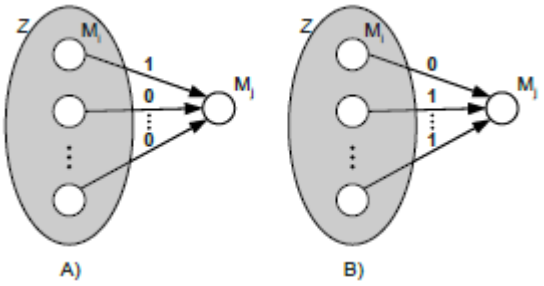
$$R_\Sigma \in R_o$$

kde R_o je množina sumárních syndromů, které mohou být obdrženy v případě, že jsou možná pouze trvalá selhání modulů, za podmínky, že počet nesprávných modulů nepřekročí hodnotu t.
Pokud je podmínka splněna, je provedena běžná diagnostika např, pomocí tabulkové metody, ale vychází se ze sumárního syndromu. Moduly označené za chybné mají buď permanentní selhání, nebo se jedná o intermitentní selhání prvého druhu.
Podmínka není splněna, je-li výsledkem sumární syndrom, který je nekonzistentní tj. obsahuje výsledky, které jsou konfliktní. Výsledek je konfliktní, pokud je v sumárním pohledu některý modul jedním správným modulem označen za správný a druhým taktéž správným za chybný. V tomto případě se obvykle další diagnostika neprovádí a výsledkem je jednoduché oznámení, že systém nemůže být správně diagnostikován. Tento případ nastává v případě, kdy v systému existují moduly s intermitentními selháními druhého a třetího druhu.

Pokud jsou však k dispozici další prostředky (časové a režijní), může procedura diagnostiky dále pokračovat, čímž lze získat další informace. Nejjednodušší možností rozšíření je zvýšení počtu opakování množiny atomických kontrol. Poté mohou být některé intermitentní moduly odhaleny jako selhávající, a to i z pohledu modulů, které tuto skutečnost ještě neodhalily, což vede k odstranění nekonzistencí a zařazení těchto modulů mezi moduly se selháním prvého druhu.
Druhá možnost je o něco zajímavější, neboť sice vyžaduje složitější prozkoumání sumárního syndromu a přináší i určitý risk (tj. pravděpodobnost nesprávného výsledku diagnostiky), nevyžaduje však provádění dalších kontrol. Základem této metody je předpoklad, že všechna zbývající neodhalená intermitentní selhání jsou třetího druhu Pravděpodobnost chybného výsledku je rovna pravděpodobnosti nesplnění tohoto předpokladu. Tento předpoklad je odůvodněný, neboť v reálných složitých systémech se tento typ intermitentních selhání vyskytuje mnohem častěji než ostatní druhy selhání.
V tomto případě (tj. pokud není splněna podmínka, je prvním krokem stanovení podmnožiny Z, do níž patří všechny moduly, které mohou být na základě sumárního syndromu R_Σ označeny jako správné. V druhém kroku je nutno ověřit konzistentnost všech výsledků atomických kontrol, jež jsou prováděny moduly z podmnožiny Z. Je tedy nutno zjistit, zda moduly z této podmnožiny stejně hodnotí moduly z podmnožiny doplňkové ($\bar{Z}$) nebo nikoliv. V průběhu zjišťování může nastat jedna ze situací znázorněných na obrázku4.18.
Situace A znázorněná na obrázku 4.18 může vzniknout z následujících příčin:

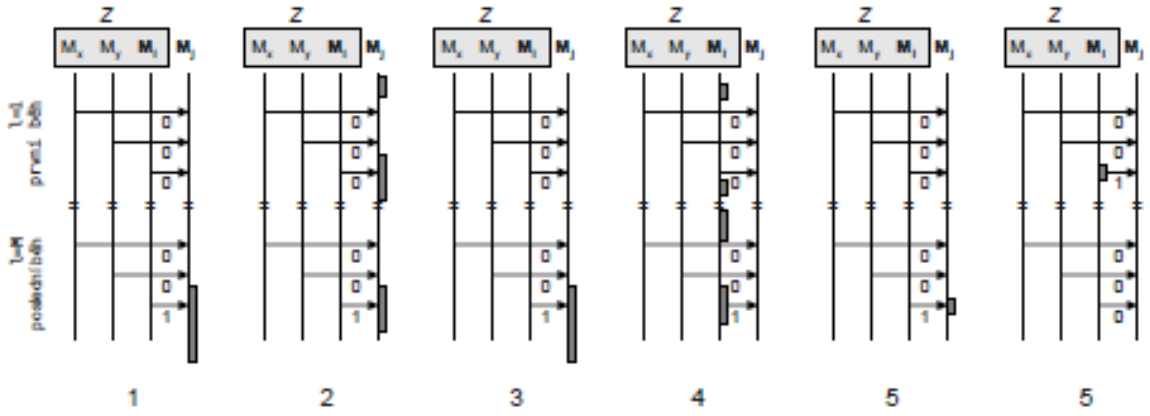
1. modul M_j selhal v okamžiku těsně před provedením poslední atomické kontroly v posledním opakování AT (zde je to kontrola provedená modulem M_i tj. τ_{ij})
2. modul M_j má intermitentní selhání druhého druhu a modul M_i je jediný z modulů, který jej zaregistroval
3. modul M_i permanentně selhal. Atomická kontrola τ_{ij} je první kontrola, která byla tímto selháním dotčena
4. modulu M_i má intermitentní selhání. Selhání bylo zachyceno pouze atomickou kontrolou τ_{ij}.
5. buď modul M_i nebo M_j má intermitentní selhání třetího druhu. Toto selhání se projevilo v průběhu atomické kontroly τ_{ij}.

Současné intermitentní selhání obou modulů nebudeme uvažovat, neboť pravděpodobnost takovéto události je velmi nízká.



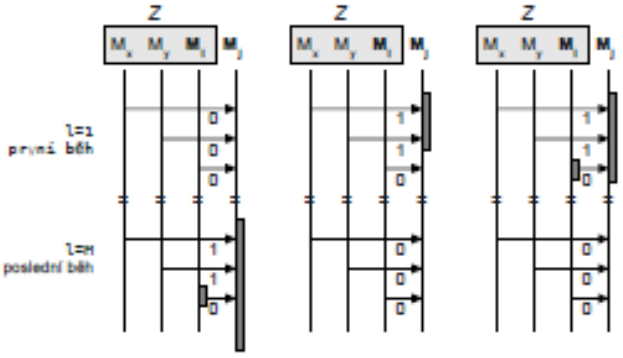
Obrázek 4.18.: Situace způsobené intermitentním selháním třetího druhu

Jednotlivé možné příčiny konfliktů jsou přehledně znázorněny v sekvenčních diagramech na obrázku4.19. V případech 2, 4, 5 existuje více možných souběhů selhání a atomických kontrol. Znázorněn je však vždy pouze jeden, resp. u pátého případu výjimečně dva.



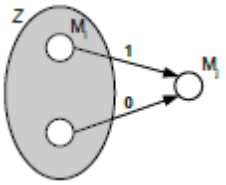
Obrázek 4.19.: Souběhy selhání a atomických kontrol v situaci A

V souladu s přijatým předpokladem budeme dále uvažovat pouze intermitentní selhání třetího druhu (viz příčina 5). Můžeme proto tvrdit, že buď modul M_i nebo M_j má krátkodobé intermitentní selhání. Takové selhání se s vysokou pravděpodobností nebude opakovat a modul bude nadále fungovat bez problémů. Proto není pro další úvahy důležité, který z obou modelů selhal. Jediným cílem je odstranění nekonzistencí v syndromu. Řešení je v tomto případě snadné, stačí zaměnit výsledek kontroly τ_{ij} z hodnoty „1“ na hodnotu „0“ .
Situace B z obrázku 4.18 je snadněji řešitelná, neboť může nastat pouze v případě, že modul M_j má intermitentní selhání, které odhalily všechny moduly z podmnožiny Z vyjma modulu M_i. Ukázky možných souběhů jsou na obrázku4.20. Řešení je jednoznačné, stačí zaměnit výsledek kontroly τ_{ij} z hodnoty „0“ na „1“; a tím zajistit konzistentní stav sumárního syndromu.



Obrázek 4.20. Souběhy selhání a atomických kontrol v situaci B

Komplikovanější situace vzniká v případě nekonzistence u dvouprvkové podmnožiny správných modulů Z (viz obrázek 4.21).



Obrázek 4.21. Situace způsobená intermitentním selháním třetího druhu (spec. případ)

Obdržený výsledek lze interpretovat buď jako situaci A (a řešit ji změnou jednoho ohodnocení na „0“) nebo jako situaci B (v tomto případě jsou ohodnocení sjednocena na „1“). Abychom mohli učinit rozhodnutí a přiklonit se k jednomu z řešení, musíme porovnat pravděpodobnosti obou situací. V případě interpretace podle situace A by se jednalo o intermitentní selhání třetího druhu modulu M_i nebo M_j. Podle druhé interpretace (situace B) musí mít modul M_i selhání druhého druhu. Protože pravděpodobnost vzniku selhání tohoto druhu je menší, je vhodnější zvolit řešení podle situace A, tj. sjednotit syndrom na hodnotu „0“ ($\tau_{ij} = 0$, volíme pozitivnější řešení).
Na konci podkapitoly ještě shrneme specifické rysy systémů s intermitentními selháními:
I. Hlavním cílem diagnostiky není identifikace modulu s intermitentním selháním, ale řešení konfliktních situací, které vznikají z důvodů specifického charakteru těchto selhání.
II. Samotná diagnostika zahrnuje několik dílčích kroků:
Krok 1: vícenásobné opakování množiny atomických kontrol a získání sumárního syndromu R_Σ.
Krok 2: ověření, zda obdržený syndrom odpovídá podmínce $R_\Sigma \in R_o$. Pokud je tato podmínka splněna, pak diagnostika probíhá stejně jako v případě permanentních selhání. V opačném případě se přechází k dalšímu kroku.
Krok 3: učení množiny Z, jenž obsahuje moduly, které lze na základě sumárního syndromu jednoznačně považovat za správné.
Krok 4**:** kontrola konzistentnosti výsledků atomických kontrol prováděných moduly z množiny Z.
Krok 5**:** vyřešení konfliktních situací.
III. Intermitentní selhání je možno rozdělit do tří druhů podle počtu opakování množiny atomických kontrol nutných pro zachycení selhání modulu.
Intermitentní selhání prvého druhu jsou odhalena již v kroku 2. Selhání druhého druhu však mohou být odhalena až po provedení kroku 3 (a to pouze některá). Selhání třetího druhu jsou tolerována. To znamená, že systém je schopen dalšího provozu bez vnějšího zásahu. Konfliktní situace způsobené intermitentními selháními třetího druhu jsou řešeny v kroku 5.

IV. Nevýhodou diagnostiky intermitentních selhání je výrazně vyšší časová složitost resp. režie systému. Proto může být velmi náročné, ne-li nemožné provádět tuto diagnostiku v průběhu provozu reálných systémů.

Kontrolní otázky:
- popište model intermitentních selhání
- charakterizujte různé druhy intermitentních selhání
- co je to sumární syndrom
- jak se provádí diagnostika intermitentních selhání

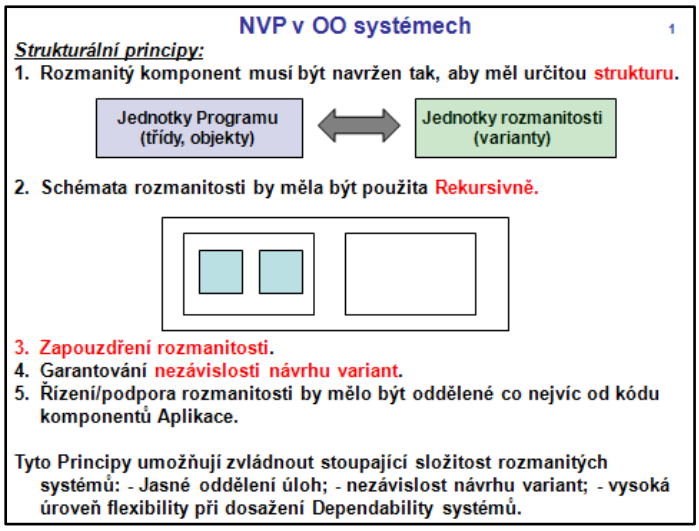
Úkoly pro samostatnou práci:
1) Prostudovat diagnostiku intermitentních selhání. Použít literaturu [1] a obsah výkladové části.
2) Seznámit se s UML nástroji pro sekvenční diagramy a prakticky ukázat jejich možnosti pro popis diagnostiky v systémech s intermitentními selháními (podle representace použité v obrázku 4.17). Použít můžete například *Enterprise Architect*, dostupný pro studenty KI).

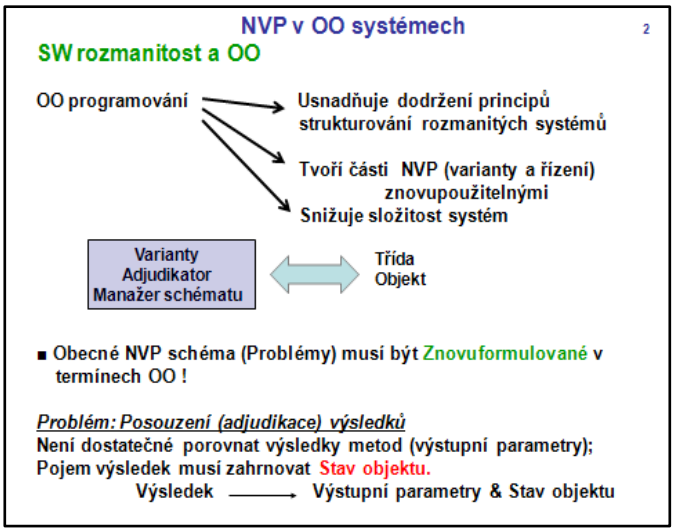
Kapitola 5. N-variantní programování a objektové orientované programování

Cíl kapitoly:
- vysvětlit výhody použití objektového programování pro N-variantní programování
- probrat specifiku použití objektů v N-variantním programování

Klíčová slova: NVP, rozmanitost, manažer, adjudikátor, varianta, obnovení, formální model

Výkladová část:
Následující prezentace vysvětluje hlavní cíly kapitoly





NVP v souběžných OO jazycích

Souběžnost je potřebná pro:
- Začít vykonání několika variant asynchronně;
- Synchronizovat varianty když budou ukončené;
- Posoudit (Adjudikovat) výsledky;
- Obnovit stav varianty.

■ Vývojáři variantních objektů používají stejnou specifikaci třídy. Varianty deklarované uvnitř RI (rozmanitě implementované) třídy. Navíc tato třída obsahuje Adjudikatorní objekt a Manažerní objekt.

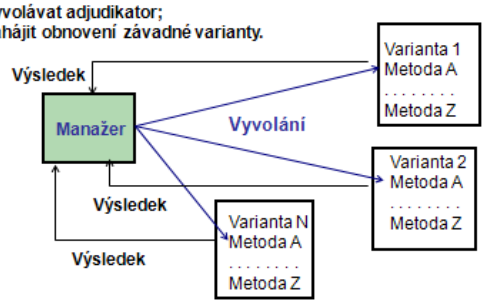
RI třída

Varianty
Adjudikator
Manažer

— Obsah RI třídy je skryt od ostatních komponentů

MANAŽER

Manažer zodpovědný za:
- začít vykonání variant paralelně;
- zachytit všechny výjimky, které mohou být produkované variantami;
- řídit vykonání variant;
- synchronizovat varianty po jejich ukončení;
- signalizovat výjimku „selhání" když je počet závadných variant příliš velký;
- garantovat konzistenci stavů variant;
- vyvolávat adjudikator;
- zahájit obnovení závadné varianty.



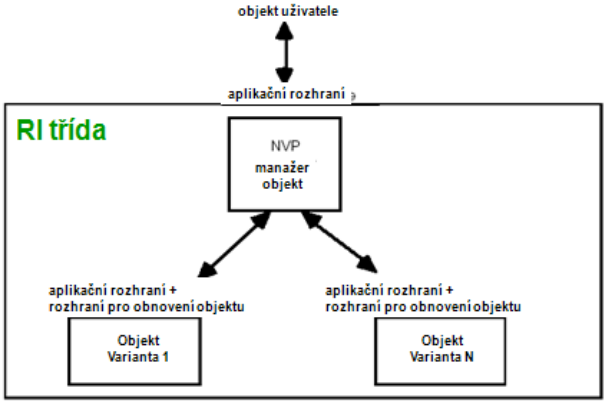
MANAŽER

■ Manažer má zajistit souběžnost a vyvolávání stejných metod ve všech variantách asynchronně.
Po ukončení vykonávání variant Manažer musí synchronizovat všechny varianty a posbírat jejich výsledky.
■ Manažer by měl být problémově nezávislý. Potřebujeme pouze „přizpůsobit" manažer nastavením jmen variantních objektů a jmen jejich metod.
■ Manažer musí být co nejvíce znovuvyužitelný!

VARIANTY

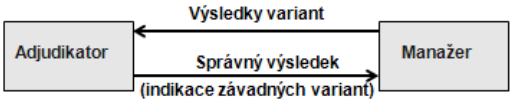
■ Variantní třídy by měly být odvozené od téže Abstraktní třídy.

Abstraktní třída ➡ slouží jako formální specifikace pro všechny programátory variant



■ Kromě aplikačních metod každý variantní objekt musí mít přídavné metody, aby umožnil obnovení varianty když její výsledek není správný.
■ Tyto metody musí být přístupné přes rozhraní variantního objektu.

ADJUDIKATOR



Adjudikator musí být problémově závislý ze dvou důvodů:
1. Manipuluje (zpracovává) reprezentaci interního stavu všech variant;
2. Výsledky variant, které musí být porovnané, jsou specifické pro každou aplikaci a mohou být jakéhokoli typu.

■ Adjudicator musí také být specifickým pro každou metodu
(Adjudikator musí mít odpovídající adjudikatornou metodu pro každou aplikační metodu)

Obnovení variant

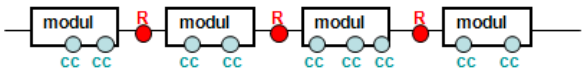
■ Menšinové výsledky variant podle NVP musí být považované za závadné.

Obecný postup:
Snažit se najít způsoby pro obnovení dat závadné varianty a používat variantu dál.

Předpoklad :
Varianty mohou mít menší počet závad, které se mohou projevit jako chyby, a po obnovení varianta může ještě dlouho sloužit.

NVP v OO systémech

Společenské obnovení (CER)
Základní myšlenka CER spočívá v tom, že stav správné varianty může být použit pro obnovení závadných variant.
- CER používá dva typy bodů, ve kterých se provádí synchronizace variant. Toto je schéma dvouúrovňového obnovení. Body křížové kontroly cc (cross-check) se používají nejen pro porovnání dat ale i pro částečné obnovení variant:
  každá varianta přijímá výsledek adjudikatoru včetně správného cc-vektoru. Tento vektor se používá pro částečné obnovení, poněvadž data ve vektoru představují podmnožinu stavu varianty.
- Každá varianta se sestává z několika modulů, které se vykonávají následně; každý z modulů má několik cc-bodů uvnitř.
- R-body jsou vložené mezi moduly a používají se pro kompletní obnovení varianty v případě, že částečné obnovení selže.



V R- bodech kompletní vnitřní stav každé varianty je mapován do prostředního formátu společného pro všechny varianty. Toto umožňuje porovnání výsledků variant a odhalení závadné varianty.

NVP v OO systémech

▼Aplikování CER v kontextu OO programování má jisté problémy:
CER popírá některé principy strukturování NVP a omezuje návrh variant (např. data v cc-vektorech musí být stejného typu; počet cc-bodů v modulech musí být stejný pro každou variantu).

Abstraktní stav varianty (AVS)
AVS je prostřední reprezentace (nebo představení) interního stavu varianty.

Problémy obnovení

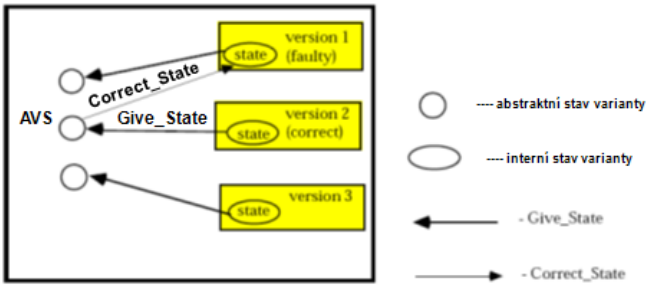
1. Vyvíjení mapujících funkcí je úkol složitý a náchylný k chybám.
2. Při zajištění obnovení variant musíme dodržovat principy strukturování rozmanitých systémů.
3. Obnovení musí zapadat do OOP.

NVP v OO systémech

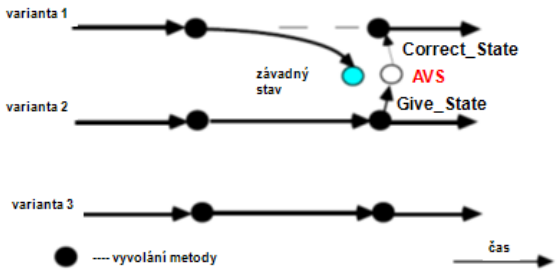
OO obnovení variant

OO obnovení variant používá dvouúrovňové odhalení chyb: porovnává buď *výstupní parametry* anebo *kompletní stavy variant.*
- První garantuje, že chybná informace nebude pašovaná navenek a výsledek systému bude správný.
- Druhé se používá, aby se snížila latentní doba chyb; stavy variant se porovnávají přes speciálně unifikované představení (reprezentaci), které se nazývá *Abstraktní Stav Varianty*.

Programátor varianty vyvíjí (vytváří) dodatečnou metodu Give_State, která vypočítá aktuální *AVS* na základě jejího interního stavu a vrátí jeho NVP manažeru.



NVP v OO systémech

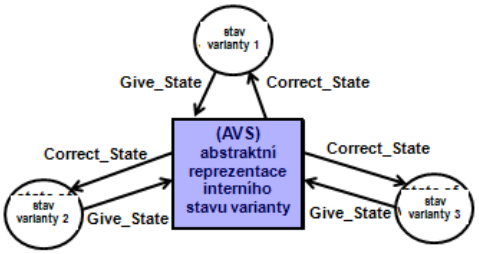


Dynamický pohled na obnovení varianty.
Stav správné varianty 2 se používá pro obnovení varianty 1.

- Problém obnovení:
  Abstraktní stav musí zahrnovat dostatečné informace pro obnovení interního stavu kterékoli závadné varianty.

NVP v OO systémech

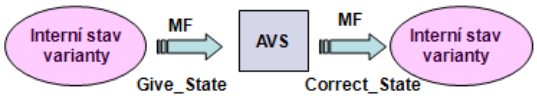
Předpokládá se, že programátor každé varianty musí implementovat dvě dodatečné metody: Give_State and Correct_State.
-První se vyvolá ve správném objektu po odhalení závadné varianty. Metoda vrátí kompletní stav této varianty v prostředním abstraktním formátu.
- Poté bude vyvolána metoda Correct_State závadné varianty. Všechna informace, která reprezentuje stav správné varianty, se předává této metodě. Tato metoda obnoví stav závadné varianty.



- AVS se používá jako prostřední formát mapujícími funkcemi (Give_State a Correct_State).

NVP v OO systémech

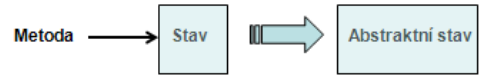
Abstraktní stavy a Mapující funkce (MF)



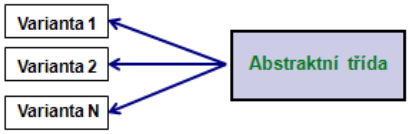
- Koncepce AVS zachycuje to, co mají společného varianty ve svých stavech.
  Stavy variant mají mezi sebou něco společného, protože jsou implementacemi téže třídy.
- Všechny varianty mají stejný průběh a proto jsou ekvivalentní a musí mít stejný Abstraktní stav.

NVP v OO systémech 15

Formální model



AVS charakterizuje konceptuální stav všech variant, které byly vyvinuty na základě téže *Specifikace* Abstraktní základní třídy.



! Všechny varianty mají stejný průběh a proto jsou ekvivalentní a musí mít stejný Abstraktní stav.

NVP v OO systémech 16

Formální model

{Vj} – množina objektů;

Sj aktuální stav objektu Vj; (neviditelný zvenku)

I_{ij} a O_{ij} jsou vstupní a výstupní parametry metody M_{ij} objektu Vj;

Pro každou metodu M_{ij} objektu Vj, výstupní parametry jsou vypočítané jako funkce F_{ij}

$$O_{ij} = F_{ij} (I_{ij} , Sj')$$

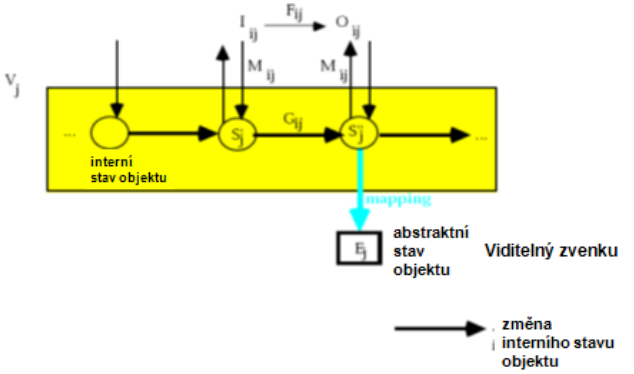
Sj' – stav objektu před vyvoláním metody M_{ij}.

Aktuální stav objektu Vj se změní z Sj' na Sj'' v důsledku vykonávání metody M_{ij} se vstupními parametry I_{ij}; toto může být popsané pomocí funkce G_{ij}

$$Sj'' = G_{ij} (I_{ij} , Sj')$$

NVP v OO systémech 17

■ Abstraktní stav Ej objektu Vj může být představen jako množina dat, která odrážejí aktuální stav objektu Vj. Tato data musí být viditelná zvenku. Data představují projekci nebo zmapované zobrazení interních dat objektu.



NVP v OO systémech 18

{Vk} – množina objektových variant, které implementují abstraktní třídu A.

Obnovení varianty V1 může být garantované, pokud existuje správná varianta Vm z {Vk} taková, že:

1. Existuje *kopírující funkce* Cm pro správnou variantu Vm, která vypočítá správný abstraktní stav Em varianty Vm (funkce Cm odpovídá metodě Give_State)

   Em = Cm (Sm)

2. Existuje *obnovovací funkce* R1 pro variantu V1 (funkce R1 odpovídá metodě Correct_State)

   S1 = R1 (Em).

■ Stav správné varianty Vm a její kopírující funkce Cm se používá pro výpočet Em. Obnovovací funkce R1 závadné varianty V1 se používá pro vypočítání jejího správného interního stavu S1.

■ Stavy S1 a Em mohou být považované za sbírku všech dat charakterizujících aktuální stav objektu a aktuální abstraktní stav.

■ Abstraktní stav a kopírující funkce musí být vyvíjeny tak, aby pro každé dvě správné varianty Vk a Vm bylo zajištěno následující:

if (Ek = Ck (Sk) & Em = Cm (Sm)) ⇨ (Ek = Em)

To znamená, že můžeme použít kteroukoli správnou variantu (která má kopírující funkce) pro obnovení kterékoli závadné varianty.

NVP v OO systémech 19

Vyvíjení abstraktního stavu objektu

■ Programátor musí analyzovat rozmanité implementace komponentu a zjistit, co společného mají data všech variant. To umožní programátoru vyvinout kopírující a obnovovací funkce pro každou variantu.

■ Existuje několik metod implementace abstraktního stavu varianty:

- Použít komponenty výhradně základního typu a udržovat je v record typu (např. real,interger, boolean, string);
- Použít interní data jedné z variant jako abstraktní stav;
- Navrhnout speciální metodologii pro vyvíjení objektových variant takovým způsobem, že jejich interní stav bude zároveň abstraktním stavem.

NVP v distribuovaných systémech

Mnohé moderní OO jazyky mají vlastnosti nutné pro programovaní distribuovaných aplikací

Problémy:

■ Jediná možnost jak použít NVP v distribuovaných systémech spočívá v umístění variant v různých uzlech, což dělá varianty viditelnými zvenku a zvyšuje šance pro pašování informace a pro zneužití varianty.

■ Předpokládá se, že budou přijata dodatečná opatření, aby se garantovalo, že pouze manažer může vyzývat varianty.

■ Dodatečné opatření musí být také přijato pro odolnost HW proti závadám v uzlech, ve kterých jsou umístěné varianty (např. použití replikace HW).

NVP v OO systémech 20

Shrnutí

- Implementace NVP v OO systémech potřebuje, aby třídy/objekty SW systémů byly zároveň i jednotkami rozmanitosti (tj. variantami).
- Rozmanitost musí být skrytá před volajícím (tj. zapouzdřená uvnitř tříd).
- Každá varianta musí být vnímaná jako oblast (kontejner) obsahující chyby.
- Varianty musí být navržené nezávisle, abychom se vyhnuli souvztažným závadám.
- NVP manažer (znovupoužitelný komponent) řídí vykonávaní variant a jejich obnovení.
- Veškerý tok informace směrující k variantě (od varianty) prochází přes manažer. To ochraňuje před pašováním chybné informace.
- Pro obnovení závadné varianty je možné použít abstraktní stav správné varianty. Pro použití abstraktního stavu varianty programátor musí vyvinout mapující funkce.

Kontrolní otázky:

- vysvětlete specifiku použití objektů pro N-variantní programování
- popište funkce manažera a adjudikátoru
- vysvětlete, jak probíhá obnovení systému

Úkoly pro samostatnou práci:
1) Prostudovat použití OOP pro N-variantní programování. Použit literaturu [1] a obsah výkladové části.
2) Vytvořit v C# jednoduchou knihovnu pro podporu n-variantního programování za využití generik a návrhového vzoru Command (s využitím generických rozhraní resp. delegátů).

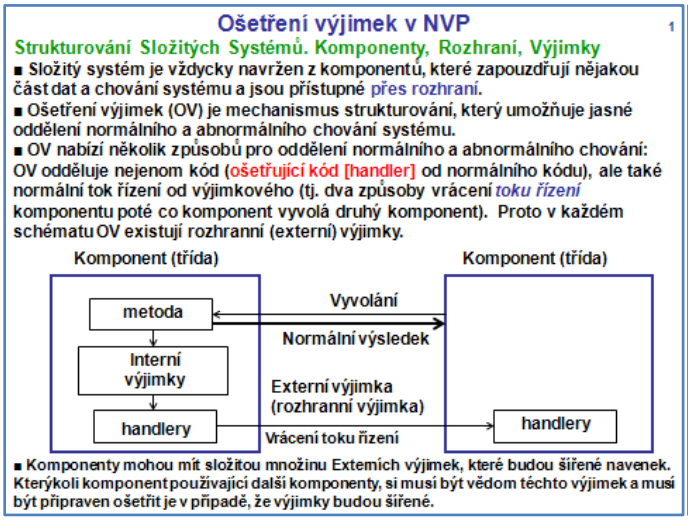
Kapitola 6. Ošetření výjimek pro N-variantní programování

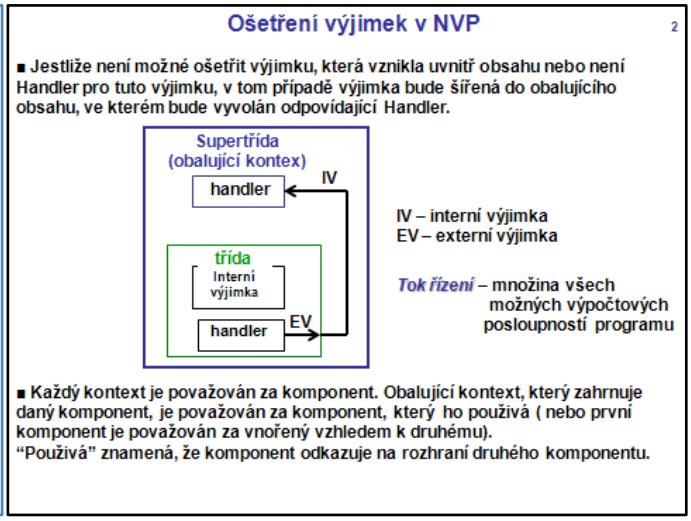
Cíl kapitoly:

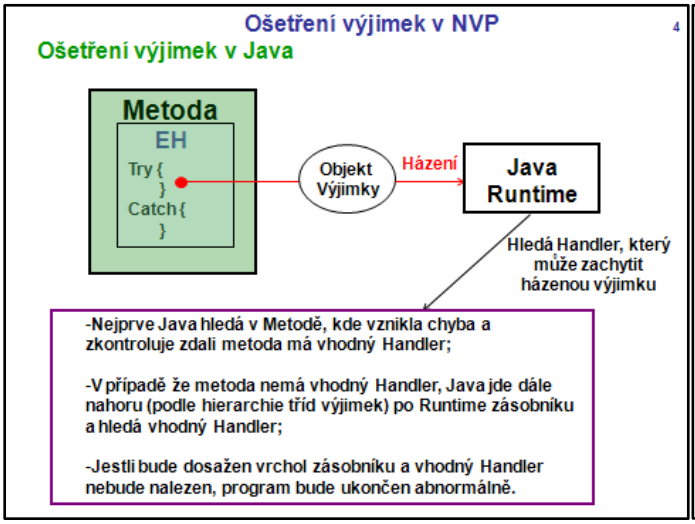
- vysvětlit model ošetření výjimek
- podrobně rozebrat interní a externí výjimky
- probrat otázku adjudikace výjimek

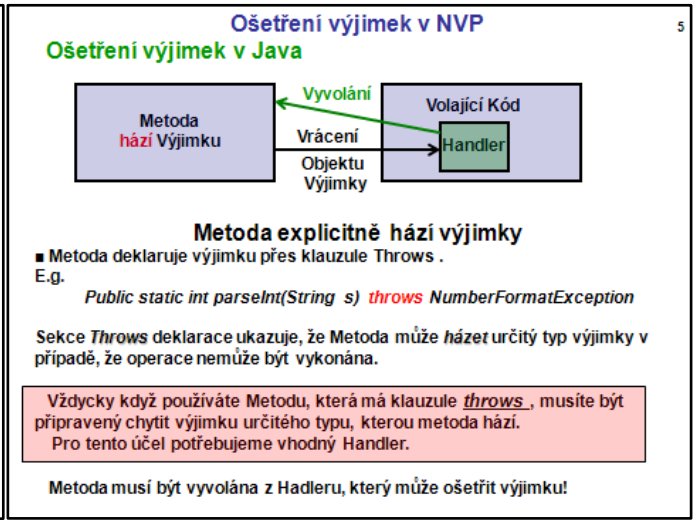
Klíčová slova: výjimka, interní a externí výjimky, ošetření výjimek, adjudikace výjimek

Výkladová část:
Následující prezentace pomůže studentům splnit úkoly samostatné práce.









Ošetření výjimek v NVP

Model Ošetření Výjimek

- V modelu ošetření výjimek každý objekt (třída) může mít několik **Externích výjimek**: E1, E2, E3,... které mohou být předané (šířené) druhým objektům.

- Objekt může mít několik **Interních výjimek** (e1, e2, e3,..): tyto výjimky jsou deklarované, vznikají a musí být ošetřené uvnitř objektu.

- Interní výjimky a jejich ošetření jsou zapouzdřené uvnitř objektu.

Vykonávání objektu může být ukončeno buď *normálně* nebo *abnormálně* (výjimečně), když externí výjimka byla signalizovaná buď z normálního kódu nebo z interního Handleru.

- Předem definovaná výjimka "Selhání" (*Failure*) se používá, když objekt není schopen vytvářet kterýkoliv přijatelný výsledek.

Omezení pro návrh objektu:

- *všechny interní výjimky musí mít handlery,*
- *všechny externí výjimky musí být specifikované v specifikaci třídy.*

- Signalizování Externí výjimky ještě neznamená, že byla chyba v průběhu vykonávání objektu. Předpokládá se, že objekt je v abnormálním stavu, když nemůže poskytnout úplně požadovaný výsledek.

Ošetření výjimek v NVP

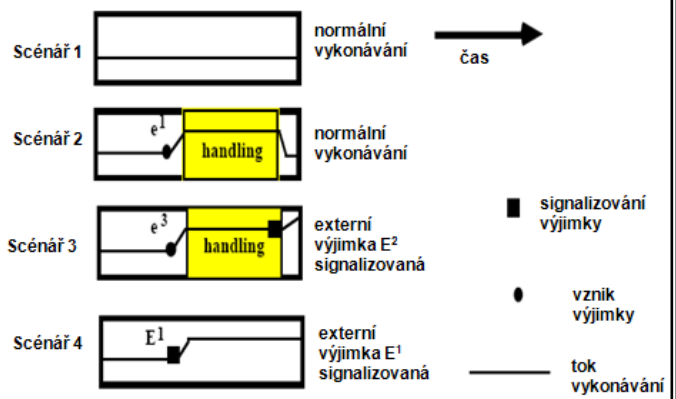
Interpretace Externích výjimek a několik způsobů jejich použití:

- ❑ Oznámit (podat zprávu), že objekt nebyl změněn ("předčasné ukončení");
- ❑ Signalizovat že objekt poskytuje částečné výsledky (tj. degenerovanou službu)
- ❑ Informovat vyvolající komponent (nebo obalující kontext) o závadách v prostředí (včetně ostatních objektů, prostředků, základních služeb apod.), které objekt odhalil, ale není schopen ošetřit;
- ❑ Oznámit že vyvolání metody není správné (např. chybné vstupní parametry) – rozhranní výjimka;
- ❑ Informovat obalující kontext, že objektové předchozí podmínky, invarianty nebo následné podmínky byli porušeny;
- ❑ Signalizovat, že v objektu se vyskytla chyba jako následek závady návrhu;
- ❑ Informovat obalující kontext, že objekt byl zanechán v prostředním stavu kvůli chybě nebo není možné vrátit zpět všechny změny.

Čtyři scénáře mohou proběhnout v průběhu vykonávání objektu:

1) Žádná výjimka nevznikla a nebyla signalizovaná; výstupem objektu je normální výsledek;
2) Vznikla interní výjimka e1 a byla úspěšně ošetřená uvnitř objektu; výstupní parametry objektu představují normální výsledek;
3) Vznikla interní výjimka e3 a odpovídající Handler se pokusil ošetřit výjimku, ale selhal, což způsobilo signalizaci externí výjimky E2 do obalujícího kontextu;
4) Externí výjimka E1 byla signalizovaná objektem okamžitě po vzniku problému bez pokusu vyřešit problém interně.

Ošetření výjimek v NVP



Ošetření výjimek v NVP

NVP a Ošetření výjimek

- Použití OV spolu s NVP umožňuje:

1. Programátoru varianty použít interní výjimky v návrhu varianty.
2. Variantám dosáhnout většinou dohodu i když několik z nich není schopno vytvářet normální výsledek.
3. Systému pokračovat v poskytování správných výsledků po vzniku a signalizaci externích výjimek v několika variantách.

Obecný popis rámce

- co jsou interní a externí výjimky v kontextu NVP?
- jak mohou RI komponenty mít rozhraní stejné jako komponenty bez rozmanitosti (když signalizují externí výjimky)?
- jak mohou RI komponenty být zahrnuty do obecné struktury systému, který používá externí výjimky?
- jak může programátor varianty použít ošetření výjimek pro vylepšení návrhu varianty? Jak může programátor používat zároveň interní i externí výjimky?
- jak mohou být posouzeny (adjudikovány) normální a výjimkové výsledky variant?

► Každá varianta má jak interní tak i externí výjimky. Interní výjimky jsou specifické pro variantu. Externí výjimky jsou stejné pro všechny varianty (tj. jsou definované rozhraním třídy) a pro celý RI komponent. (RI komponent jako celek)

Ošetření výjimek v NVP

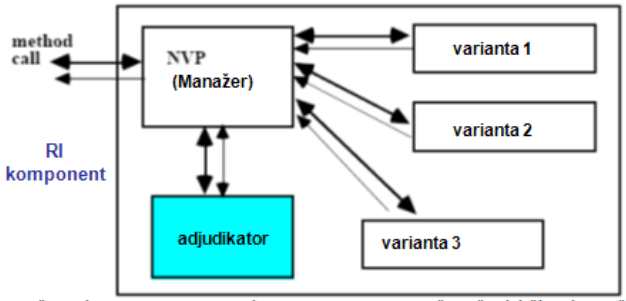
► Vykonávání každé jednotlivé varianty vždy probíhá podle jednoho ze 4 scénářů.

Po ukončení všech variant NVP manažer (kontrolor) vyvolává Adjudikator, který musí mít *rozšířenou* funkčnost pro zpracování (interpretace) většinových výsledků ve všech možných scénářích. Adjudikator rozhoduje, které varianty produkovaly chybné výsledky (což znamená, že tyto varianty jsou považované za závadné, protože mají SW závady).
-Jestli většina vydala (poskytuje) stejný normální výsledek, Adjudikator oznámí tento výsledek (podává zprávu);
-Jestli většina signalizuje stejnou výjimku, Adjudikator oznámí tuto výjimku. Potom manažer signalizuje tuto výjimku v prostředí komponentu;
-Jestliže varianta je v menšině (výsledek varianty patří menšině), její výsledek je považován za chybný a sama varianta za závadnou;
-Jestliže většina není, bude signalizovaná předem definovaná výjimka "Selhání" V tomto případě musíme ignorovat tento RI komponent a vyloučit jej z dalšího použití pokud nebude provedená speciální opravná procedura.
Tento přístup dovoluje použít externí výjimku "Předčasné ukončení" (Abort).

Rozmanitě implementovaný komponent

- V RI objektu, funkčnost a rozhraní všech komponentů jsou rozšířené, aby se přizpůsobilo (zařadilo) ošetření výjimek (v porovnání s obvyklým NVP).
- RI komponent vypadá při pohledu zvenku jako normální komponent, rozmanitost je skrytá.

Ošetření výjimek v NVP



Manažer má k dispozici rozhraní RI komponentu. Manažer přebírá řízení pokaždé když je vyvolána metoda RI komponentu. Poté manažer vyvolává varianty a adjudikator.

- Manažer má běžnou funkčnost co se týče variant a adjudikatoru. Jinak, manažer má speciální funkčnost: ***sbírání výjimek*** signalizovaných variantami, jejich předávání adjudikatoru, signalizování adjudikovaných výjimek volajícímu komponentu (bude-li potřeba). Manažer je problémově nezávislý.
- Rozhraní RI komponentu a rozhraní variant jsou stejné. Tím pádem RI a varianty mohou šířit pouze stejné externí výjimky.

Ošetření výjimek v NVP

12

Interní Výjimky Variant

Každá varianta Vi může mít interní výjimky: e^{i1}, e^{i2},.., e^{ik}, a odpovídající handlery: eh^{i1}, eh^{i2},.., eh^{ik}, deklarované uvnitř Vi.
Tyto výjimky nemohou být šířené navenek.
Handler eh^{ij} buď uspěje v ošetření výjimky e^{ij} (v tom případě varianta vrací normální výsledek (vrací řízení a výstupní parametry) anebo varianta signalizuje externí výjimku z Handleru eh^{ij}, která bude předána NVP manažeru.
Interní výjimky jsou specifické pro každou variantu a musí být vyvinuté nezávislými programátory.

Externí Výjimky Variant

Rozhraní RI objektu a variant obsahují stejné výjimky: E1, E2, E3,.., Em.
Každá varianta buď vytvoří normální výsledek nebo signalizuje externí výjimku.
- Předpokládá se, že externí výjimka "Selhání" bude vytvořená variantou a předaná manažeru když bude překročen timeout (tj. vyprší čas určený pro vykonání metody varianty).
- Manažer a adjudikator zpracují výjimku "Selhání" zvláštním způsobem. Jestli stav varianty není obnoven, varianta nemůže být dále použitá a její výsledky nejsou důvěryhodné.

Ošetření výjimek v NVP

13

Adjudikace Výjimek

- Externí výjimky variant nemohou být předané přímo do kontextu vyvolávače: výjimky musí být nejdříve adjudikované.
- Poté, co všechny varianty budou ukončené, manažer předává jejich výsledky adjudikatoru.

Pro tento model je to rozšířený adjudikator, který umí zacházet z výjimkovymi výsledky signalizovanými variantami. Cílem rozšířeného adjudikatoru, stejně jako obyčejného, je nalezení většinových výsledků.

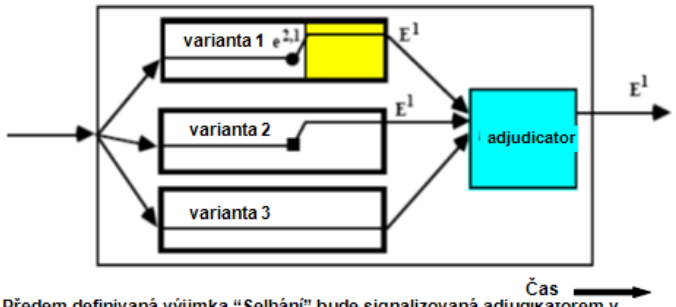
! - Navíc, adjudikator provádí odhalení chyb (tj. zjistí varianty, které vytvořily menšinové výsledky a proto jsou považované za závadné).

Chybný výsledek	Výjimkový výsledek
Výsledek varianty je považován za chybný když tato varianta byla nalezena v menšině.	Výjimkový výsledek nemusí znamenat chybu, a varianta, která vytvořila tento výsledek, nemusí být závadnou. Výjimkový výsledek, který patří většině, bude předán dále do obalujícího kontextu a bude zpracován na vyšší úrovni.

- Varianty V1 a V2, které produkovaly výjimku E^1, nejsou v chybném stavu. Kromě situace když většina variant signalizuje výjimku "Selhání".

Ošetření výjimek v NVP

14



- Předem definivaná výjimka "Selhání" bude signalizovaná adjudikatorem v případě, že není konsenzus mezi výsledky variant. Každá varianta může signalizovat výjimku "Selhání". V tom případě postup adjudikatoru je následující: jestli většina variant signalizuje "Selhání" adjudikator bude signalizovat výjimku "Selhání". Jinak, výsledek závadné varianty (která patří menšině) bude ignorován (tj. maskován).
- Rozšíření adjudikatoru nezpůsobí zvětšení jeho složitosti, protože existuje konečný počet externích výjimek v kterékoli RI třídě a jejich porovnání nepotřebuje složité operace.

Ošetření výjimek v NVP

15

Příklady (ukazují jak rozšířený adjudikator pracuje v systému s třemi variantami):

(E1, E2, E1) ⇒ E1

- (E1, E2, E3) ⇒ Selhání
- (E1, E2, Normal2) ⇒ Selhání
- (Normal1, Normal2, E1) ⇒ Selhání
- (Failure, Failure, Normal) ⇒ Selhání

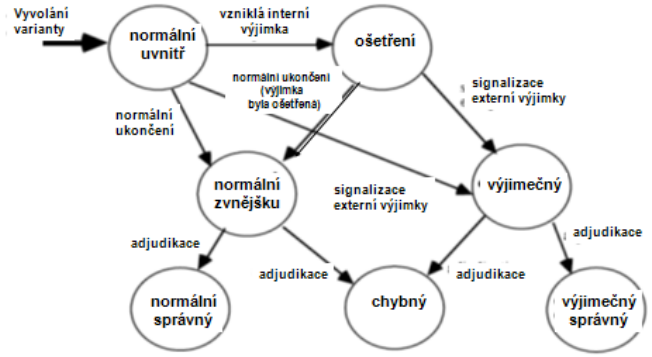
Externí výjimky RI třídy. Šíření výjimek

- Manažer signalizuje adjudikovanou výjimku do obalujícího kontextu (volajícího komponentu), který musí zpracovat výjimku. Aby se umožnilo zpracování výjimky, volající komponent musí mít Handlery pro všechny výjimky, které mohou být signalizované.
- Výjimka "Selhání" RI komponentu je konceptuálně identická s výjimkou "Selhání" variant: tato výjimka bude signalizovaná když není možné poskytnout jakýkoliv důvěryhodný výsledek a komponent je zanechán v nekonzistentním stavu. Na úrovni RI komponentu toto znamená, že buď není většina nebo varianty signalizují výjimku "Selhání".

Ošetření výjimek v NVP

16

Diagram přechodů stavů varianty



V průběhu vykonávání varianta může být buď ve stavu "normální uvnitř" nebo ve stavu "ošetření". Po ukončení varianta může být buď ve stavu "normální zvnějšku" nebo ve stavu "výjimečný". Následující adjudikace zjistí, zdali tento stav je správný nebo chybný.

Ošetření výjimek v NVP

17

✓ Programátor každé varianty rozhoduje, které interní výjimky použít a jak je ošetřit.

✓ Všichni programátoři musí dodržovat specifikaci Externích výjimek.

✓ Specifikace každé externí výjimky musí zahrnovat:
- přísný popis podmínek (výjimkové podmínky), při kterých výjimky budou signalizovány;
- popis stavu, ve kterém objekt bude zanechán poté co bude signalizována výjimka (následné podmínky).

Jedině toto může garantovat že:
- stejné výjimky budou vždycky signalizované všemi správnými variantami
- existuje výjimka pro každou abnormální situaci (pro varianty a pro RI objekt)

✓ Výjimka může být signalizovaná variantou pouze poté, co je garantováno, že interní stav varianty splňuje odpovídající následné podmínky.
✓ Stavy všech správných variant musí být stejné, když signalizují stejnou výjimku.

Programátoři variant musí zabezpečit: Výjimkové podmínky; Následné podmínky a Podmínky zabezpečující, že stavy všech správných variant jsou stejné (identické).

Kontrolní otázky:

- popište model ošetření výjimek v NVP
- charakterizujte externí výjimky
- jak probíhá adjudikace výjimek

- popište výjimkové podmínky (s použitím příkladů)

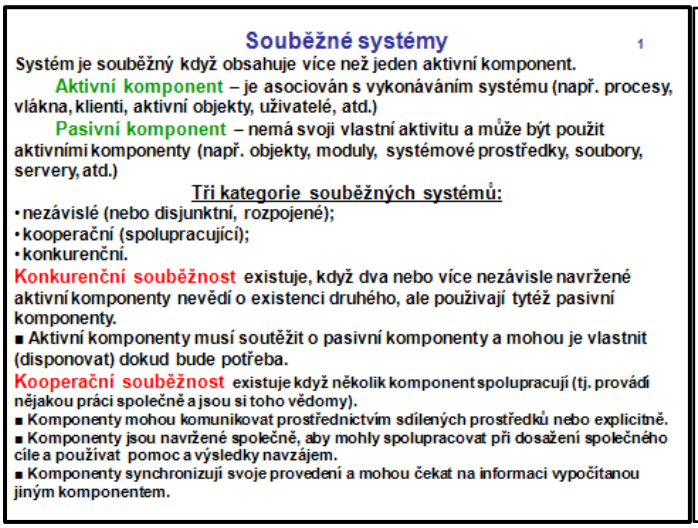
Úkoly pro samostatnou práci:
1) Prostudovat ošetření výjimek v N-variantním programování. Použit literaturu [1] a obsah výkladové části.
2) V programovacím jazyce C# vytvořit program pro ošetření výjimek v NVP.

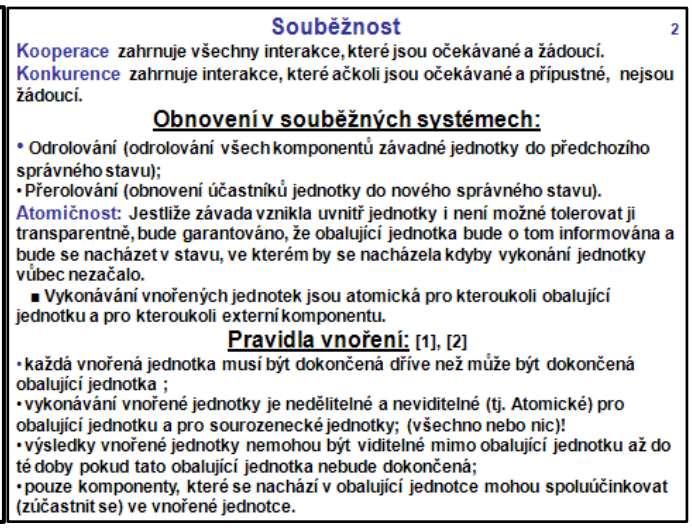
Kapitola 7. Konkurenční a spolupracující souběžné systémy

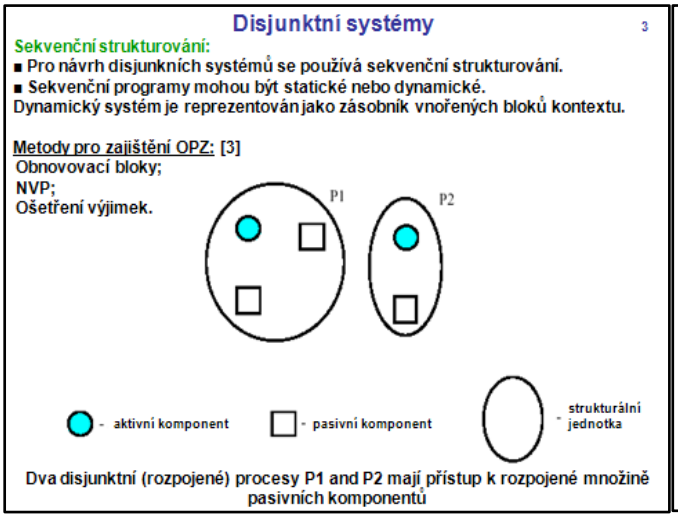
Cíl kapitoly:
- charakterizovat tři kategorie souběžných systémů
- podrobně rozebrat kooperační a konkurenční souběžnost
- probrat metody slučující kooperační a konkurenční souběžnost

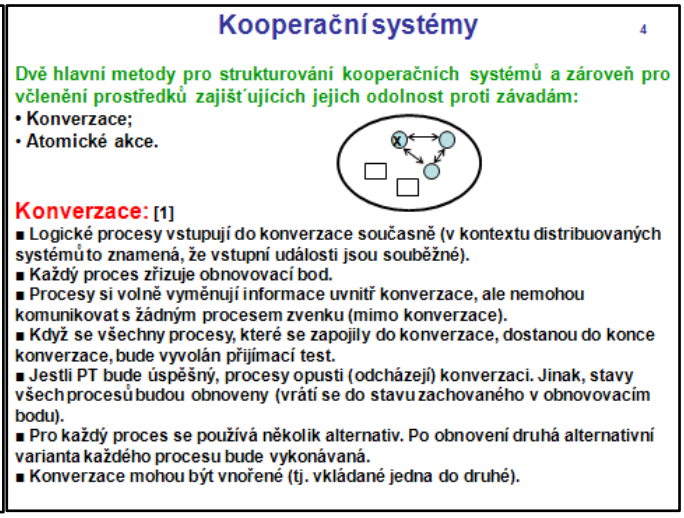
Klíčová slova: souběžnost, aktivní komponent, pasivní komponent, transakce, AKIT vlastnosti

Výkladová část:
Následující prezentace pomůže studentům splnit úkoly samostatné práce.

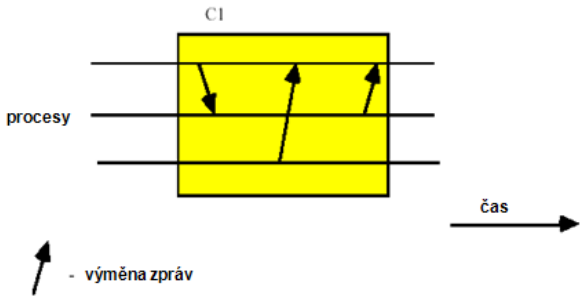








Kooperační systémy



Konverzace C1: tři účastníci

Kooperační systémy

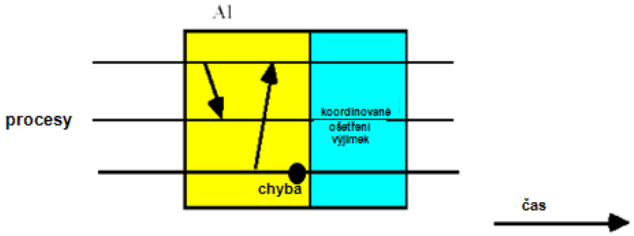
Atomické akce: [4], [5]
Byly vyvíjeny jako zevšeobecnění konverzace.

Atomické akce používají
- Odrolování
- Přerolování
- nebo jejich kombinace.

Odrolování je identické tomu, které se používá v konverzaci.
Přerolování se spoléhá na *Mechanismus Ošetření Výjimek* a může včlenit (začlenit) dodatečný mechanismus pro vyřešení četných výjimek, které vznikly souběžně v několika komponentech.

Kooperační systémy

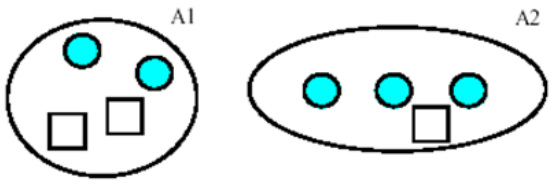
- Ke každému účastníku AA (procesu) je přiřazen svůj odpovídající Handler; Handlery, které určené pro ošetření stejné výjimky, budou vyvolané ve všech účastnících.



Atomické akce A1: chyba byla odhalená, všichni tři účastníci jsou zapojeni do koordinovaného obnovení (ošetření výjimky)

Kooperační systémy

Systém sestává ze dvou Atomických akcí



Dvě atomické akce A1 a A2 . Akce A1 má dva aktivní komponenty a akce A2 má tři aktivní komponenty

- Každá akce má několik procesů, které navzájem komunikují. Vykonání akce A1 a A2 je disjunktní (není souvislé).
- Pasivní komponenty uvnitř akce jsou jak prostředky pro kooperace tak i data reprezentující stav akce/systému nebo procesu.
- Všeobecně, za výsledky vykonání akce jsou považované jak stavy pasivních tak i aktivních komponentů.

Konkurenční systémy

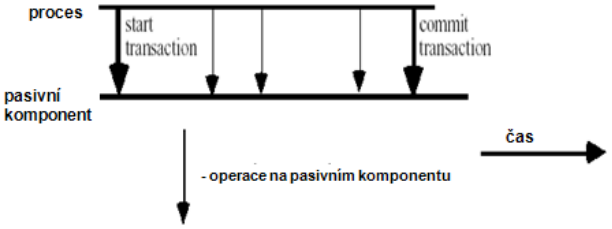
Je vyvinuto mnoho typů jednotek pro sestavení struktury konkurenčních systémů. Tyto jednotky mohou být tříděné, mohou odpovídat různé úrovni systémů a jsou určené pro různé oblasti použití.
První schémata byla vyvinuta pro sdílené prostředky v multi-uživatelských OS a DB.
Jedna z prvních metod pro strukturování konkurenčních systémů byla metoda založená na Atomické Transakci.
Tato metoda popisuje obecná pravidla pro strukturování konkurenčních aplikací a pro uvedení atomických jednotek v programovacím rozhraní.

Atomické transakce: [6, 7, 8, 9]
Čtyři vlastnosti (*AKIT vlastnosti*) strukturálních jednotek:
- Atomičnost;
- Konzistentnost;
- Izolovanost;
- Trvanlivost.

Konkurenční systémy

Atomická transakce spoléhá na tři standardní operace:
- Start;
- Abort (předčasné ukončení);
- Commit (uložit do paměti).

Operace Start a Commit markují (označují) hranice Atomické Transakce.



- Atomická transakce začíná aktivním komponentem, který pak provádí několik operací na pasivním komponentu a commit transakce (ukončuje transakce uložením stavu pasivního komponentu).
- Všechny pasivní komponenty mohou být přístupné několika transakcemi současně ale AKIT vlastnosti těchto transakcí jsou garantovány Podporou Transakce.

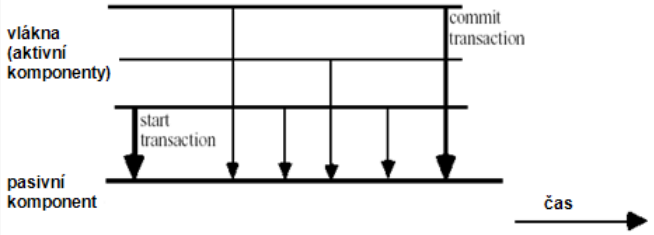
Konkurenční systémy

Multi-vláknové transakce [10]

MvT dovolují několika aktivním komponentům (vláknům) zúčastnit se stejné (téže) transakce a provádět operace na stejných objektech (pasivních komponentech) společně.

- Účastníci stejné transakce nespolupracují jakýmkoliv způsobem !
- Vykonávání vláken není synchronizované v rámci (v mezích) MvT.

(Pro toto schéma obnovení znamená Abort transakci).



MvT začíná aktivní komponentem; pak několik aktivních komponentů provádí operace na pasivních komponentech a jeden z nich commit (ukončuje) transakci

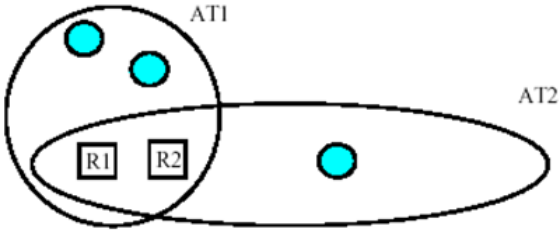
Konkurenční systémy

Atomické transakce jsou určené pro zajištění odolnosti pouze proti HW závadám.

- OPZ je zajištěná prostřednictvím odrolování stavů objektů (pasivních komponentů).

- ***Atomická transakce*** nezačleňuje žádné rysy
  - pro zajištění odolnosti proti SW závadám nebo závadám prostředí;
  - pro zajištění obnovení aktivních komponentů.

Konkurenční systémy



Dvě atomické transakce, AT1 a AT2. AT1 má dva aktivní komponenty a AT2 má jeden komponent

- Výsledky provedení AT jsou reprezentované stavy pouze pasivních komponentů.

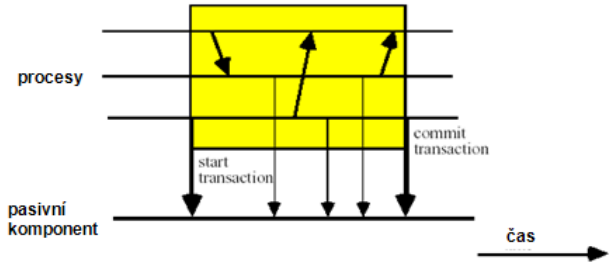
- AT garantuje, že soupeřící komponenty (tj. komponenty z různých transakcí) budou vykonávané tak jako by byly disjunktní (rozpojené).

Metody slučující kooperační a konkurenční souběžnost

Rozšířené konverzace [13], [14], [15]

Systém sestává z:

- (takzvaných "konverzačních") kooperačních komponentů, které synchronizují své obnovení (tj. vrácení ke kontrolním bodům) pomocí mechanismu konverzačního typu.
- serverových komponentů, které jsou podpořené mechanismem Atomické Transakce.



Rozšířená konverzace EC1 s třemi účastníky

Metody slučující kooperační a konkurenční souběžnost

Rozšířené konverzace:

Serverové komponenty přijímají požadavky (ve formě zpráv nebo dálkového volání procedury) a provádějí transakční operace START, ABORT a COMMIT.

- Tyto systémy jsou strukturované rekurzivně použitím vnořených konverzací. Vnořené konverzace mohou mít přístup k transakčním objektům (pasivním komponentům).

- Podpora rozšířené konverzace:
  - podpora garantuje, že konverzace budou správně manipulovat pasivními objekty: operací starts, abort a commit (provedené transparentně pro programátory kooperačních systémů) dodržují striktní pravidla, která garantují správné chování kombinovaného systému;
  - podpora zajišťuje koordinované obnovení (Odrolování) všech komponentů zapojených do rozšířené konverzace.

- RK zajišťuje odolnost proti SW závadám spolupracujících komponentů.
- RK garantuje, že konverzace správně manipulují transakčními objekty.

Závěr: RK = Konverzace + Atomická Transakce

Metody slučující kooperační a konkurenční souběžnost

Koordinované Atomické akce (KA akce): [12], [16]

KAA = Atomická akce + Atomická transakce

KA akce je jednotný obecný přístup k strukturování komplexní souběžné aktivity. Schéma umožňuje také obnovení četných interaktivních objektů v distribuovaných OO systémech.

- KA akce poskytuje konceptuální rámec pro zacházení jak s různými typy souběžnosti (kooperační a konkurenční) tak i s prostředky zajišťujícími odolnost proti závadám. KAA rozšiřuje a integruje dva komplementární koncepty - Atomická akce a Transakce.

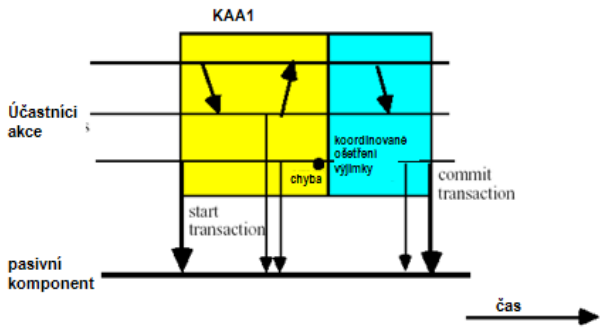
KA akce mají vlastnosti obou:
Atomické akce se používají pro řízení kooperační souběžnosti a pro implementace koordinovaného obnovení komponentů.
Transakce se používají pro udržování konzistence sdílených prostředků v přítomnosti selhání a konkurenční souběžnosti.

- Každá KAA je navržená jako stylizovaná Multi-vstupní procedura s rolemi, které mohou být aktivované účastníky KAA (aktivními komponenty) a které mohou kooperovat uvnitř KAA.

Metody slučující kooperační a konkurenční souběžnost

19

Koordinovaná Atomická akce



Koordinovaná Atomická akce CAA1 se třemi účastníky, kteří provádějí koordinované obnovení (přerolování)

Koordinovaná Atomická akce

20

KA akce může být dokončena:

- Normálně
  - buď když žádná chyba nevznikla (chyba nebyla odhalena)
  - nebo po úspěšném obnovení
- Abnormálně (výjimečně) – když výjimka „selhání" byla šířená do obalující akce

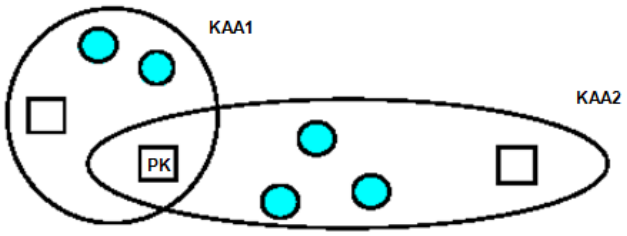
■ Externí (transakční) objekty (tj. pasivní komponenty) mohou být použité souběžně několika KAA takovým způsobem, že:
- informace nemůže být pašovaná mezi nimi (tj. mezi KA akcemi)
- kterákoli posloupnost operací na těchto objektech, která je umístěná uvnitř závorek KAA (start – commit), má AKIT vlastnosti – vzhledem k ostatním posloupnostem.

■ KA akce vypadá jako Atomická transakce pro ostatní prostředí.

■ Stav KA akce je reprezentován množinou stavů lokálních a externích objektů.

Koordinovaná Atomická akce

21



Dvě KA akcí, KAA1 a KAA2, jsou sestavené z kooperačních komponentů a soupeří o pasivní komponent PK

■ Účastníci těchto jednotek spolupracují přes pasivní komponenty, které patří výhradně jedné z těchto jednotek.
■ V KAA existují dva typy pasivních komponentů:
- Lokální (používá se pro spolupráci účastníků);
- Sdílené několika jednotkami (mají AKIT vlastnosti).
■ Výsledky vykonávání těchto jednotek jsou reprezentované stavy jak aktivních tak i pasivních komponentů.

22

metoda	souběžnost	závady	obnovení
konverzace (1976)	kooperační	závady návrhu	odrolování (rozmanitost SW)
atomické akce	kooperační	závady prostředí, závady návrhu	přerolování (výjimky) a odrolování (rozmanitost SW)
atomické transakce (mid-60ies)	konkurenční	závady HW	odrolování (opakování)
rozšířené konverzace (1991)	kooperační a konkurenční	závady návrhu	odrolování (rozmanitost SW)
koordinované atomické akce (1995)	kooperační a konkurenční	závady prostředí, závady návrhu, závady HW	přerolování (výjimky) a odrolování (rozmanitost SW, opakování)

Porovnání různých dynamických metod strukturování SW systémů

Implementace

23

Atomické akce mohou být aplikované pro strukturování: [6]
- systémů pracujících v reálném čase;
- řízení procesů;
- telefonických spínacích systémů;
- elektronického vybavení letadla

a pro zajištění jejich odolnosti proti závadám.

■ Bylo navrženo několik úspěšných aplikací pro *Konverzace* a *Atomické akce*, které byly implementované v praktických systémech, v ukázkových příkladech a reálných příkladech:
- námořní příkazový a řídicí systém; [18]
- zjednodušený dopravní řídicí systém; [19]
- model antiraketového obranného C3 systému; [20]

Vlastnosti Atomické transakce byly vnesené do mnohých programovacích jazyků, do distribuovaných a OS, atd.[7,8] Například, všechny systémy řízení DB se hodně spoléhají na Atomické Transakce.

Srovnání Atomické akce a Atomické Transakce

24

❑ V dnešní době Atomická transakce (ATr) se používá více než Atomická akce:
- programátoři si zvykli, že je lepší pracovat s programem když mají pocit, že všechny prostředky jsou v jejich vlastní dispozici;
- podporující SW (včetně OS a ATr) garantuje, že programátor při vyvinuti SW bude pracovat ve vlastním prostředí a nemusí se starat o koordinaci a kooperaci s ostatními komponenty souběžného systému;
- hodně úsilí bylo věnovano vývoji Transakčních systémů. Je těžké změnit tento trend (směr vývoje), tj. stereotyp chování.

Nicméně použití ATr všude by mohlo vést ke zvýšení nákladů a je zavádějící

Proč potřebujeme kooperace:
- souběžné aktivity může potřebovat výměnu informace nejenom přes pasivní AKIT komponenty, ale může potřebovat synchronizaci jejich provedení a může mít společný cíl;
- je hodně důkazů, že závady SW se stávají dominujícím faktorem. Proto potřebujeme metody strukturování SW, které umožňují zajistit odolnost proti všem druhům závad.

Nové složité distribuované a souběžné systémy (Internetové aplikace, rozšířené pracovní toky, řídicí systémy, moderní telefonické spínací systémy, systémy řízení elektrických sítí , multi-robot dílny, komplexní middlewarové služby, atd.) vyžadují kooperace a metody pro strukturování, které kombinují kooperace a konkurence.

Kontrolní otázky:

- definujte tři kategorie souběžnosti

- definujte co je to aktivní a pasivní komponent systému
- pojmenujte vlastnosti atomické transakce
- popište multi-vláknovou transakci

Úkoly pro samostatnou práci:
1) Prostudovat kooperační a konkurenční souběžnost. Použit literaturu [1] a obsah výkladové části.
2) Prostudovat a na praktickém příkladě presentovat implementaci transakcí na úrovni jazyka C# (třída *TransactionScope*). Vyzkoušejte je pro implementaci transakcí mezi dvěma databázemi.

Kapitola 8. Koordinované atomické akce

Cíl kapitoly:
- vysvětlit co je Koordinovaná Atomická Akce (KAA)
- probrat hlavní problémy návrhu KAA
- uvést příklady praktického využití KAA

Klíčová slova: vlastnosti KAA, vnoření, konkretizace, ošetření výjimek

Výkladová část:
Následující prezentace vysvětluje hlavní cíly kapitoly

KA – akce 1

KA akce – je sjednocené schéma pro koordinaci komplexní souběžné aktivity a pro podporu obnovení četných interaktivních objektů v distribuovaných OO systémech.

- **KA akce zajišťuje koncepční rámec pro zacházení s různými druhy souběžnosti a dosažení odolnosti proti závadám pomocí rozšíření a integrování dvou komplementárních koncepcí: konverzace a transakce.**
- **KA akce zahrnuje strategii zacházení s závadami HW, SW a prostředí a zajišťuje koordinované obnovení interakčních objektů.**

KA akce 2

KA akce – je mechanismus pro koordinaci multi-vláknových interakcí a pro zajištění konzistentního přístupu k objektům v přítomnosti konkurenční souběžnosti a potenciálních závad.

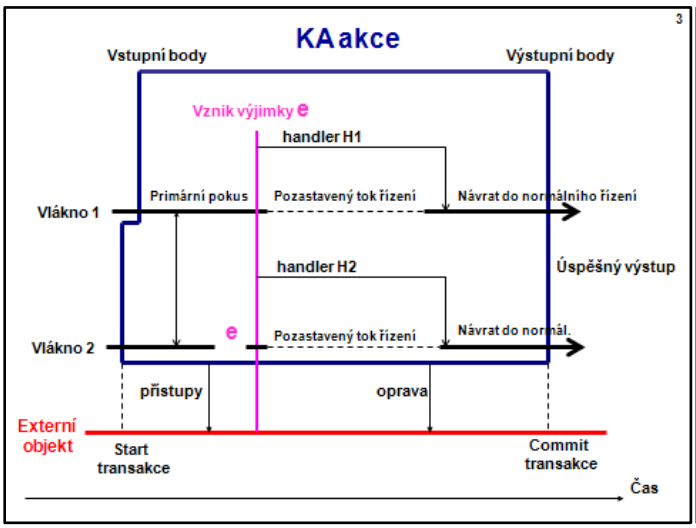
Odrolování **: KAA zajišťuje obnovovací čáru, která koordinuje obnovovací body objektů a vykonávání vláken, která se zúčastní KAA, abychom se vyhnuli „domino“ efektu.**
Přerolování **: KAA zajišťuje efektivní metody koordinování Handlerů.**

Přijímací Test **určuje, jestli výsledky KA akce jsou správné.**

Provedení KAA
- **Vlákna, která se zúčastní KAA, mohou vstoupit do KAA asynchronně, ale jejich výstup z KAA musí proběhnout synchronně.**

V případě, že bude odhalena chyba uvnitř KAA, bude provedeno odpovídající obnovení (přerolování nebo odrolování). Obnovení musí být vyvolané společně, abychom dosáhli vzájemně konzistentního závěru (řešení).

KA akce 3



KA akce 4

Příklad (bankovní aplikace)

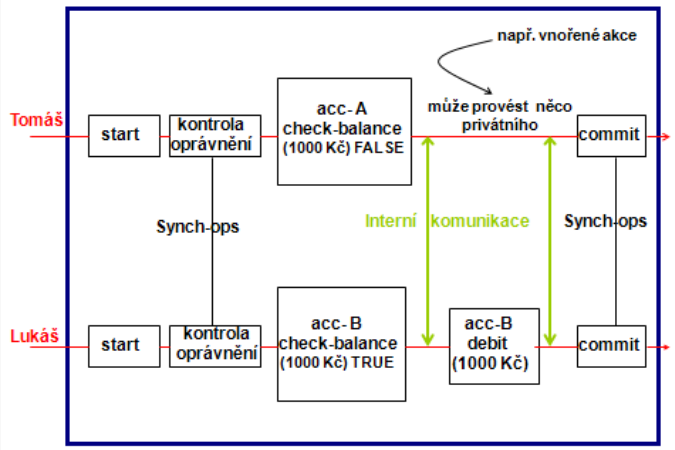
- **Systém převodu fondů je implementovan jako množina interaktivních objektů spolu se systémem zpracovaní Transakce, který zodpovědný za zajištění konzistentního přístupu k těmto objektům.**
- **Objekt = bankovní konto;**

Interní stav objektu = aktuální zůstatek konta;

Operace na objektu:
- **credit (dát na úvěr)**
- **check-balance (kontrola zůstatku)**
- **debit (výběr)**
- **read-balance, atd.**

Vlákna = Klienti systému převodu fondů (Tomáš a Lukáš).

KA akce DEBIT



KA akce

Vlastnosti KA akce

- KA poskytuje mechanismus pro vykonávání skupiny operací na množině objektů atomicky.

  Tyto operace jsou vykonávané kooperativně jednou nebo více rolemi. Role jsou vykonávané paralelně uvnitř KAA.
- Abychom provedli KAA, musí se skupina vláken sejít a dohodnout vykonávat role KAA souběžně (jedno vlákno pro každou roli).
- Četná vlákna uvnitř KAA komunikují navzájem přes lokální objekty, které jsou zcela interní pro KAA a jsou používané pro podporu kooperační souběžnosti.

  Tímto způsobem vlákna uvnitř KAA mohou koordinovat svou souběžnou aktivitu a dohodnout se na množině operací, které chtějí provést na pasivních objektech.

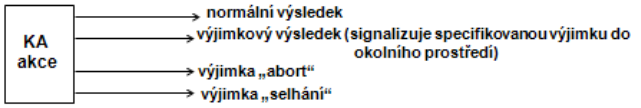
  Tyto pasivní objekty jsou považované za externí vzhledem k KAA a mohou být zpřístupněny dalšími KA akcemi, které budou soutěžit o tyto pasivní objekty.

  Pro zajištění korektnosti a prevence pašování informace KAA se musí chovat jako Transakce vzhledem k externím objektům.

KA akce

Vlastnosti KA akce

- Je uznáno, že operace prováděné vlákny uvnitř KAA mohou být neúspěšné a existuje možnost použít různé prostředky pro zredukování pravděpodobnosti vzniku selhání.
- Požadovaný efekt vykonávání KAA je popsán přijímacím testem, který je strukturován v termínech normálního výsledku a řady výjimkových (degradovaných) výsledků.
- Efekty vykonání KAA se stávají viditelnými pouze když přijímací test bude úspěšný.



- Každé vlákno zapojené do KAA musí obdržet indikace (označení) výsledku:
  - normální výsledek
  - výjimkový výsledek
  - výjimka „abort“
  - výjimka „selhání“

Všechna vlákna se musí dohodnout na vytvořenem výsledku.

KA akce

Vlastnosti OPZ

- v průběhu provedení KAA v jednom z vláken může vzniknout výjimka
  - tato výjimka musí být ošetřena lokálně vláknem.
  - jestli výjimka nemůže být ošetřena lokálně, výjimka musí být šířená do ostatních vláken KAA.
- výjimky mohou vzniknout v několika vláknech přibližně současné
  - proces vyřešení výjimek bude vyvolán, abychom se dohodli na výjimce, která bude šířená a ošetřená uvnitř KAA.
- jakmile dohodnutá výjimka bude šířená do všech vláken, bude použit odpovídající mechanismus obnovení.

Jestli není možné dosáhnout ani normálního výsledku ani výjimkového výsledku pomocí obnovovacího mechanismu, musí být KAA Abortováno a její efekty musí být zrušené.

Prostředky (opatření) pro zajištění OPZ zahrnují i efekty, které jsou následky provedení KAA.

Odrolování musí obnovit stav všech objektů, které se zúčastnily KAA (včetně externích objektů)

Přerolování musí zajistit, že všechny objekty budou zanechány v přijatelném stavu.

KA akce

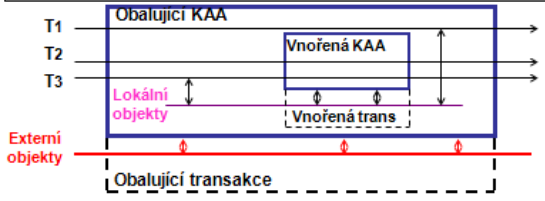
Vnoření

- Vnořené KAA mohou být použité pro strukturování provedení obalující KAA.
- Efekty vnořené KAA se stávají viditelnými pro obalující KAA hned po jejím ukončení. .
- Vlákna uvnitř KAA koordinují svoje aktivity přes množinu lokálních objektů, které jsou interní pro KAA.

Vzhledem k vnořeným KAA tyto sdílené objekty jsou externí objekty. Takže pravidla pro interakci s nimi musí být odlišná.

Jestli vnořená KAA vzájemně působí (interaktuje) s objekty, které jsou pro ni externí, musí být toto působení atomické vzhledem k ostatním vnořeným KA akcím a k obalující KA akci.

Vnořená KAA musí vstoupit do vnořené Transakce s každým objektem, který je pro ni externí i když tyto objekty nejsou externí pro obalující KA akci.



KA akce

Vnoření

- Protože ne všechna vlákna uvnitř KAA musí být zapojena do vnořené KA akce, je velká pravděpodobnost, že vznikne konkurenční souběžnost mezi vnořenými KA akcemi a vlákny obalující KA akce.

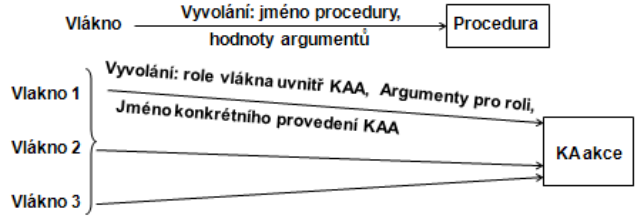
Vnořené transakce mohou zabránit možné zasahování do vnořené KAA a garantují serializovanost.

- Vnoření dovoluje KA akcím podpořit jak kooperační tak i konkurenční souběžnost na různých úrovních abstrakce.

Konkretizace

- Je nutné identifikovat konkrétní provedení KA akce, protože každé provedení zahrnuje více než jedno vlákno.

Provedení KAA se bude konat pouze když určený počet vláken vyvolá KA akci (každé vlákno specifikuje zvláštní roli uvnitř téhož provedení akce).



Mechanismus pro identifikaci konkrétního provedení KAA:

Tento mechanismus používá některý druh tokenu. Kterékoli vlákno, mající přístup k tomuto tokenu, se může zúčastnit konkrétního provedení akce pod podmínkou, že specifikuje svou roli uvnitř akce.

Konkretizace

Vyvolání KA akce:

1. Některý agent (vlákno) zodpovědný za identifikaci potřebnosti vyvolání konkrétní KA akce, vygeneruje token, který bude reprezentovat (označovat) konkrétní provedení akce.
2. Agent (vlákno) zodpovědný za komunikaci předává hodnotu tokenu různým vláknům, která budou vykonávat role v akci.
3. Jednotlivá vlákna používají token ve vyvolávacím příkazu KAA, aby ukázala, že se hodlají zúčastnit konkrétního provedení KA akce. Každý takový vyvolávací příkaz musí jmenovat konkrétní roli v KA akci, kterou vlákno bude provádět, a specifikovat argumenty, které musí být předány této roli.

Problémy návrhu KAA:

- Koordinace vláken;
- Ošetření výjimek a rozhodnutí (řešení problému četných výjimek);
- Koordinovaný přístup k externím objektům;
- Zajištění odolnosti proti závadám návrhu.

1. Koordinace vláken

• Synchronizace • Dezerce vláken • Globální verzus lokální PT

Synchronizace

Vlákna mohou vstoupit do KAA asynchronně, ale musí být synchronizována při výstupu z akce.

Synchronizace výstupu znamená dosažení dohody o úspěšnosti nebo selhání akce, což ovlivňuje jejich následný postup: buď Commit akce nebo se pokusit o obnovení poté, co chyby byly odhalené PT.

Vlákna musí opustit KAA synchronně. Je možné, že signalizují výjimku do okolního prostředí.

Synchronizace vláken při výstupu

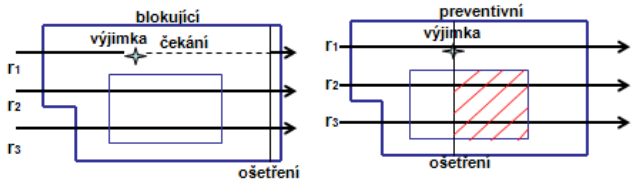
Centralizované systémy	Distribuované systémy
-Mechanismy programování souběžných systémů (monitors, semaphores, atd.) -Schémata logického synchronizování	- Kauzální pořadí doručení - Spolehlivá skupinová komunikace

Ošetření výjimek

- ❑ Blokující
- ❑ Preventivní

■ V blokujícím schématu každá role buď bude ukončena při dosažení konce akce nebo v průběhu provedení akce selže, o čemž svědčí vzniklá výjimka.
Ale ostatní role zapojené do akce budou informovány o vzniklé výjimce pouze, když budou ukončené a připravené přijmout informace o stavech druhých rolí.

■ Preventivní schéma nečeká na ukončení ostatních rolí a využívá některých rysů jazyka, aby přerušilo všechny role, když v jedné z nich bude odhalená chyba.



Mechanismy jako timeouter a run-time error checks (kontrolery chyb doby provádění) mohou zvýšit efektivitu blokujícího schématu a snížit čas čekání pomocí předčasného odhalení buď chyby anebo abnormálního chování role akce.
V preventivním schématu není čekání (mrhání časem), ale toto schéma vyžaduje přerušení vláken, což není zajištěno mnohými programovacími jazyky.

Dezerce vláken

Tento problém vzniká když jedna z roli nebude aktivovaná nebo jedna z roli nedosáhne konce akce.

■ Dvě formy dezerce: *vstupní* a *výstupní*.

Vstupní dezerce musí být ošetřená obalující KA akcí. Takže akce, která odhalila vstupní dezerci, musí být abortováná a odpovídající výjimka „Selhání“ musí být šířená do obalující KA akce s nadějí, že dezerční vlákno bude zapojeno do obnovení na vyšší úrovni.

Globální verzus Lokální PT

Požadovaný efekt vykonání akce je popsán PT (tj. co očekáváme od vykonání akce). Je to Globální PT pro celou akci.

Ve mnohých konverzačních schématech návrháři nepoužívají Globální PT a dávají přednost Lokálním PT pro každého účastníka konverzace.

Globální PT má několik předností:

1. Umožňuje zkontrolovat stav KA akce;
2. Dovoluje zkontrolovat stavy externích objektů;
3. Dovoluje zkontrolovat výstupní parametry (když je KAA má);
4. Dovoluje zkontrolovat komplexní společné podmínky pro stavy akcí, rolí, externích objektů a výstupních parametrů.

■ Abychom provedli Globální PT, je nutné synchronizovat všechna vlákna a zajistit, že všechny externí objekty se nachází v konzistentním stavu.

2. Ošetření výjimek a Rozhodnutí (vyřešení problému četných výjimek)

■ V průběhu provedení KA akce je možné, že několik vláken odhalí chyby a signalizují výjimky současně.
Takové chyby musí být ošetřené koordinovaným způsobem.

■ Strom výjimek se používá, aby uspořádal výjimky, které mohou být signalizované v průběhu KA akce.
Předpokládá se, že Handler určený pro ošetření výjimky vyšší úrovně, je schopen ošetřit kteroukoli výjimku nižší úrovně.

■ V OO systému výjimky jsou tříděné podle typu, což umožňuje snadno vytvořit strom. Pomocí stromu výjimek je možné vyřešit souběžné výjimky.

■ Po rozhodnutí souběžných výjimek stejné handlery budou vyvolány v každé roli a proběhne přerolování.

■ Handlery musí být přiřazené ke každé roli. Takže když vlákno vstupuje do akce, vstupuje také do odpovídajícího výjimkového kontextu.
Některá nebo všechna vlákna mohou posléze vstoupit do vnořených KA akcí. Takové vnoření způsobí vnoření výjimkových kontextů.

každá role ← *přiřazená* — množina handlerů

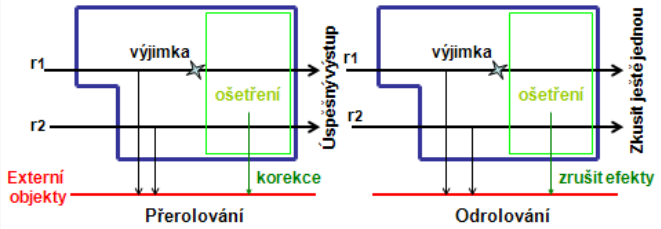
KA akce

17

2. Ošetření výjimek a rozhodnutí

- Výjimky mohou být šířené přes vnořené kontexty cestou, která odpovídá řetezci vnoření KA akcí.

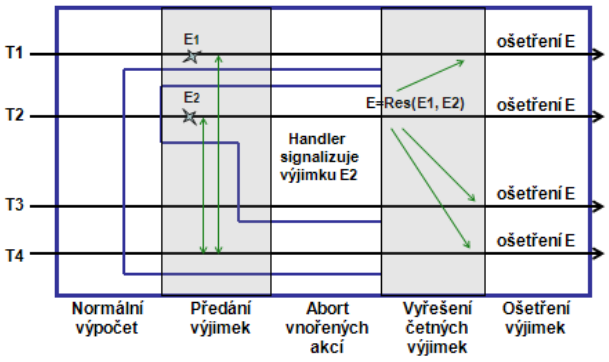
- Handlery jsou zodpovědny za koordinované obnovení a buď úspěšně ukončují KA akce (možná, že signalizují výjimkový výsledek) nebo se pokouší provést Abort (tj. zrušit efekty akce). Když se operace Abort nepodaří bude signalizována výjimka „Selhání".



KA akce

18

2. Ošetření výjimek a rozhodnutí



Ošetření souběžných výjimek a rozhodnutí

KA akce

19

3. Koordinovaný přístup k externím objektům

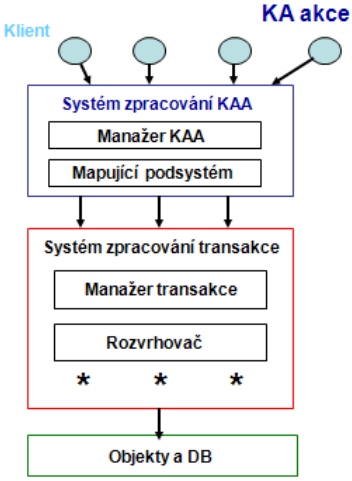
V praxi uspořádání operací na externích objektech → Pomocí speciálně navrženého *mapujícího* mechanismu, který musí být přidán na existující transakční mechanismus

- Pořadí provedení operací je určené i pořadím specifikovaným každým vláknem i komunikací mezi vlákny.

Tyto komunikace musí být vzaty v úvahu při určování korektního mapování množiny vnořených KA akcí na odpovídající množinu vnořených transakcí.

KA akce

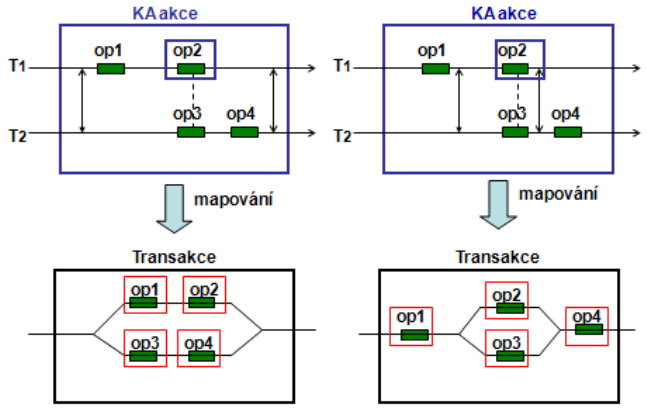
20



CA actions

21

3. Koordinovaný přístup k externím objektům



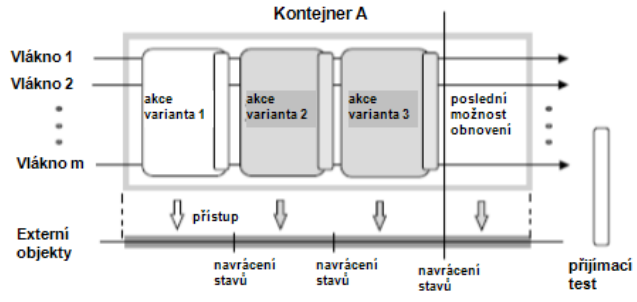
Mapování vnořených KA akcí na vnořené transakce

KA akce

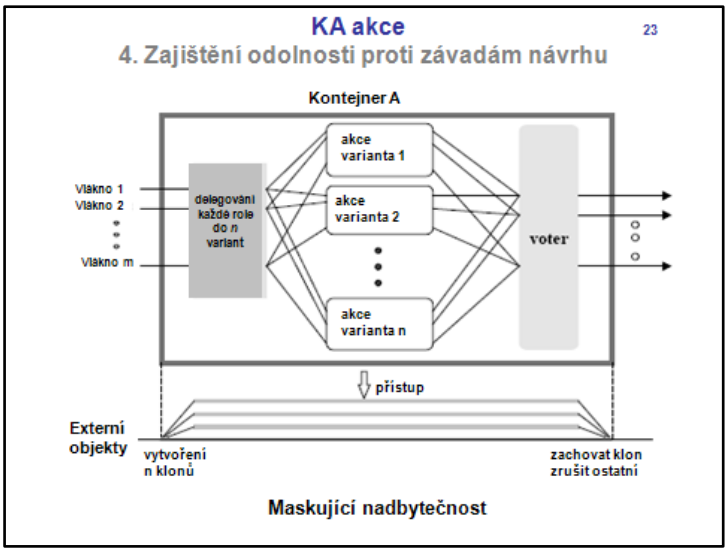
22

4. Zajištění odolnosti proti závadám návrhu

Pro danou KA akce *A* a její specifikaci je vyvinuto několik variant nezávisle. Akce *A* může sloužit jako kontejner. Uvnitř kontejneru množina vnořených KA akcí slouží jako SW varianty. Každá varianta zvlášť zajišťuje skutečnou funkčnost akce A a společně varianty poskytují nadbytečnost nutnou pro vypořádání s SW závadami.



Dynamická nadbytečnost



Kontrolní otázky:

- definujte problémy návrhu KAA
- jak probíhá ošetření výjimek v KAA
- vysvětlete problém dezerce vláken
- popište jak může být zajištěna odolnost KAA proti závadám

Úkoly pro samostatnou práci:
Prostudovat koordinované atomické akce. Použit prezentaci ve výkladové části.

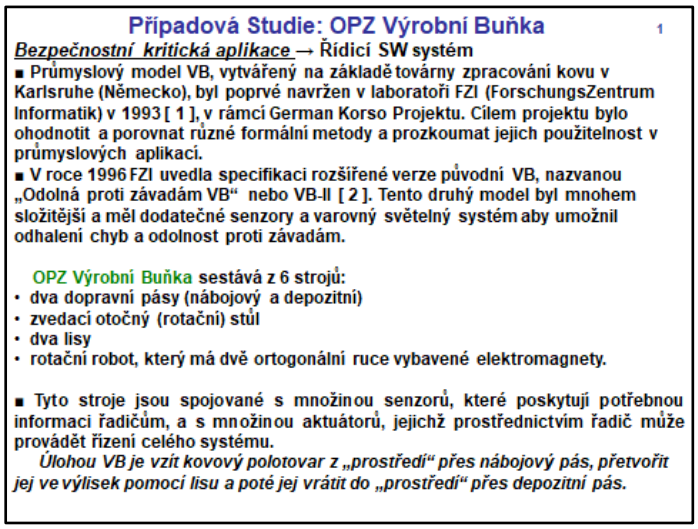
Kapitola 9. Příklad použití koordinované atomické akce

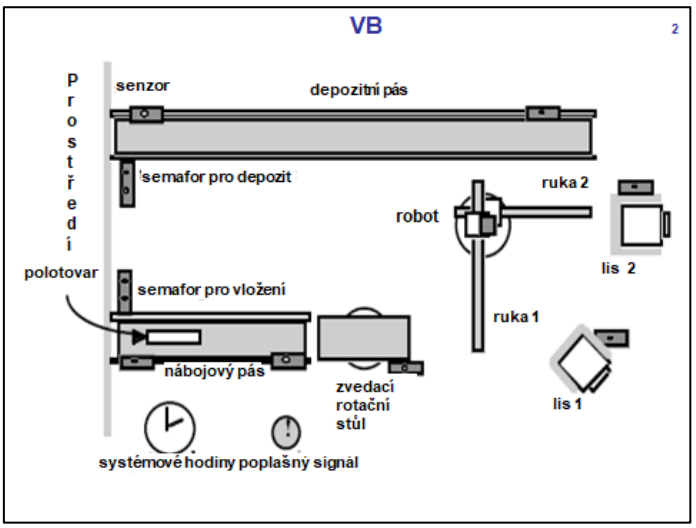
Cíl kapitoly:

- podrobně rozebrat příklad použití KAA (Případová studia).
- specifikovat problémy použití KAA při návrhu řídícího programu

Klíčová slova: případová studia, řídící program, odhalení selhání, návrh KAA

Výkladová část:
Následující prezentace vysvětluje příklad použití KAA





Produkční cyklus pro každý polotovar:

1) Když dopravní světlo pro vložení polotovaru signalizuje zelenou, polotovar může být přidán do nábojového pásu z prostředí (např. zásobovatelem předlisků);

2) Nábojový pás transportuje polotovar do stolu.

3) Stůl se točí a stoupá do pozice, kde magnet robota je schopen uchopit polotovar;

4) Ruka1 robota zvedne polotovar a umístí jej do neobsazeného lisu, buď do lisu 1 nebo do lisu 2;

5) Vybraný lis zpracuje polotovar;

6) Ruka2 robota sundá slisovaný výlisek z lisu a umístí jej na depozitní pás;

7) Když semafor pro depozitní pás svítí zelenou, výlisek může být posunut dopředu a přenesen do prostředí (kde může být použit kontejner pro skladování výlisků).

Správný Řídicí Program musí splňovat určité požadavky:

Bezpečnostní požadavky:

- pohyblivost strojů musí být omezená;
- srážkám (střetnutí) strojů musí být zabráněno;
- polotovary nesmí být umístěné mimo bezpečné zóny (tj. nábojový pás, stůl, lis, depozitní pás);
- dostačující vzdálenost musí být udržována mezi polotovary.

Životnost:

Každý polotovar, umístěný (vnesený) do VB přes nábojový pás, musí posléze opustit VB přes depozitní pás a musí být slisován jedním ze dvou lisů.

Odhalení selhání a nepřetržitost provozu:

Vzniklé selhání (jedno z velkého počtu definovaných selhání) musí být odhalené. Když se vzniklé selhání netýká lisu, systém musí být zastaven v bezpečném stavu.
Po opravě VB, která obvykle vyžaduje účast (činnost) opraváře, systém musí být schopen pokračovat v provedení operací, přičemž začne z tohoto bezpečného stavu.

Systémové hodiny, Stopky a poplašné signály

- VB-II zajišťuje globální systémové hodiny, které poskytují aktuální čas kdykoliv. Na základě těchto hodin řídicí program může implementovat několik Stopek dohlížejících na individuální procesy (např. na pohyb nábojového pásu).
- VB-II zajišťuje také poplašné signály pro hlášení selhání komponentů uživatele VB.
- Řídicí program je nutný pro zapnutí (spuštění) poplašného signálu vždycky, když selhání bude odhaleno. Vypnutí poplašného signálu provádí operátor (obsluha) po opravení stroje, který selhal.

Definice selhání:

Předpoklad 1: Systémové hodiny, dva semafory a mechanismus poplašných signálů jsou bezporuchové (tj. správné a nemohou selhat).

Předpoklad 2: Hodnoty senzorů, aktuátorů a hodin jsou vždycky přenášené správně (tj. bez výpadků a chyb).

Předpoklad 3: Selhání nemohou vést k tomu, že stroj překročí určité omezené pozice; v nejhorším bude stroj zastaven automaticky.

Předpoklad 4: Veškerá selhání senzoru jsou indikována hodnotami senzoru.

Předpoklad 5: Veškerá selhání aktuátorů způsobí zastavení stroje.

Pro daný stroj jsou zvažována následující selhání:
- Selhání senzorů;
- Selhání aktuátorů;
- Ztracení polotovaru nebo přilepení (uvíznutí) polotovaru.

Kompletní rozbor selhání v [3].

Selhání Robota

Selhání senzoru:
Existují tři senzory, které jsou přiřazené robotu. Každý senzor vrací jednu z několika předem definovaných hodnot, které označují pozice ruky robota a rotační pozice.

Selhání aktuátoru:
Existují tři druhy aktuátorů přiřazených robotu. Každý druh má svoje vlastní režimy selhání:

1) Režimy selhání aktuátorů, které stáhnou zpět ruku, zahrnují:
   - není odezva (tj. nemůže se pohybovat);
   - nepředvídané zastavení pohybující se ruky.

   Tato selhání mohou být odhalena pomocí kontrolních hodnot senzorů.
2) Režimy selhání aktuátorů, které zapíná a vypíná magnet ruky, zahrnují:
   - není odezva (např. ruka nemůže zachytit nebo upustit polotovar);
   - nepředvídané zachycení nebo upuštění.

   Tato selhání mohou být odhalena jenom pomocí kontrolních hodnot senzorů dalších strojů, které působí společně s robotem.
3) Režimy selhání aktuátorů, které otáčejí robotem, zahrnují:
   - není odezva (tj. nemůže se otáčet);
   - nepředvídané zastavení otáčejícího se robota.

   Tato selhání mohou být odhalena okamžitě pomocí kontrolních hodnot senzoru, který signalizuje rotační polohu robota.

Ztracení polotovaru:
Tento typ selhání může být odhalen jenom na základě kontroly hodnot skupiny senzorů, které patří různým strojům, působícím společně s robotem.

Selhání lisu

Selhání senzoru:
Existují čtyři senzory přiřazené každému lisu. Jeden signalizuje, je-li polotovar v lisu (tzv. senzor polotovaru). Ostatní tři signalizují polohu stroje.
- Selhání senzoru polotovaru mohou být odhalena pomocí kontroly, jestli už ruka robota předala/vzala polotovar lisu.
- Selhání senzoru, který signalizuje polohu lisu, může být odhalena pomocí stopek a dalších senzorů lisu. Informace stopek umožňuje změřit čas pohybu stroje.

Selhání aktuátoru:
Režimy selhání aktuátorů, které posouvají lis, zahrnují:
- není odezva (tj. nemůže se pohybovat);
- nepředvídané zastavení pohybujícího se stroje.

Tato selhání mohou být odhalena pomocí informací senzorů, které signalizují polohu lisu, a hodnot stopek.

Ztracení nebo přilepení polotovaru:
Tato selhání mohou být odhalena jenom na základě informací senzorů, které signalizují, zda je polotovar v lisu.

Způsoby odhalení selhání

Assertion statements(Tvrzení) je běžný způsob odhalení selhání.
Tvrzení – prohlášení, že v určitém bodě programu platí určitá podmínka.

Například. Poté co řídicí program poslal řídicí příkaz robotu a žádal ho umístit polotovar do lisu 1, hodnota senzoru signalizujícího, že polotovar je ve stroji, musí být zkontrolovaná pomocí tvrzení. Jestli senzor vrátí 0, což ukazuje, že polotovar není ve stroji 1, musí být signalizovaná odpovídající výjimka.

Existuje několik možností, které mohly způsobit tuto výjimku:
1) Polotovar mohl být ztracen;
2) Ruka1 robota mohla selhat při upuštění (umístění) polotovaru;
3) Senzor lisu 1 mohl selhat při signalizování, že polotovar byl umístěn do lisu 1.

- Ve většině případů, když vznikne selhání a výjimka bude signalizována, VB bude prostě zastavena v bezpečném stavu.

- Selhání senzorů, které signalizují polohu lisu, a selhání aktuátorů lisu mohou být odhalena pomocí Trvzení a jednoznačně identifikována pomocí Stopek.

Řídicí program (s použitím KA akcí)

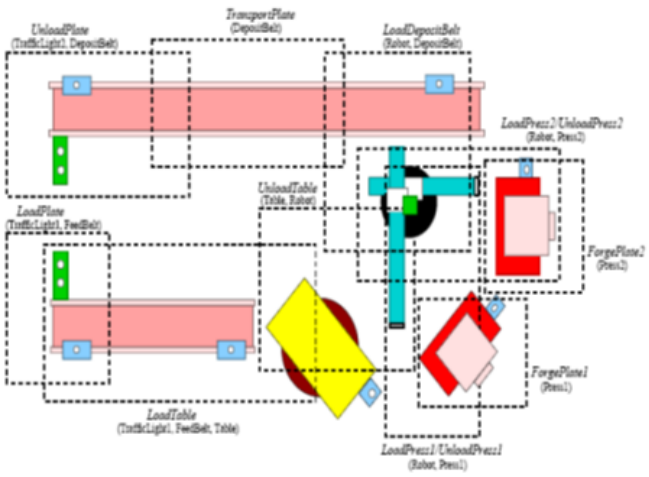
Řídicí program zahrnuje 12 hlavních KAA: každá akce řídí jeden krok zpracování předlisku a typicky zahrnuje předávání polotovaru mezi dvěma stroji.

Každý stroj se může pohybovat pouze „uvnitř" KAA.

V řídicím programu je šest souběžných vláken (provedení), která odpovídají šesti strojům:
- nábojový pás;
- stůl;
- robot;
- lis 1;
- lis 2;
- depozitní pás.

▪ Veškeré pohyby strojů jsou provedené „uvnitř" KA akcí. Stroje zahrnuté do každé KA akce budou zastaveny dříve než opustí akci. Takže když stroj není pod řízením akce, bude stroj bez pohybu (stacionární).

Polotovar je navržen jako Externí Objekt vzhledem ke KA akcím.



▪ Obvykle jedna role KA akce vezme polotovar jako vstupní argument a stroj odpovídající této roli předává polotovar další roli, která jej vrátí jako výstupní argument.

▪ Průnik mezi KA akcemi ukazuje, že tyto KA akce nemohou být provedené současně.

▪ Vzájemné vyloučení KAakcí garantuje, že ani polotovar ani stroj nemohou být zapojené do více než jedné akce (najednou) ve stejnou dobu. Takže ani polotovary ani stroje se nemohou střetnout (srazit se).

▪ Každý HW stroj je asociován s řadičem stroje (tj. s vláknem provedení). Řadič je zodpovědný za dynamické určení posloupnosti akcí, kterých se stroj může zúčastnit.

Návrh KA akcí

Předchozí a následné podmínky:

- Akce může začít jenom tehdy, když její předchozí podmínky jsou validní.

▪ Pro danou KA akci se tyto podmínky používají aby se zajistilo, že provedení akce bude splňovat požadavky systému (bezpečnostní a OPZ).

Příklad: KAA *LoadPress1 (ZatíženíLisu1)*

Tyto podmínky mohou být zkontrolované pomocí hodnot příslušných senzorů a stavů příslušných aktuátorů.

(např. abychom zkontrolovali, zdali je robot vypnutý, je nutné ověřit že všechny příslušné aktuátory jsou pozastavené (tj. v bezpohybovém stavu); abychom zkontrolovali, že ruka1 a ruka2 robota jsou vtažené, je nutné zkontrolovat hodnoty senzorů, které signalizují polohu (pozice) ruky).

předchozí podmínky	následné podmínky
robot je vypnutý	robot je vypnutý
polotovar v ruce 1	ruka 1 je prázdná
obě ruce jsou vtažené	obě ruce jsou vtažené
robot je v jednom z definovaných úhlů	úhel robotu: ruka1 směrem k lisu1
lis 1 je vypnutý	lis 1 je vypnutý
lis 1 je prázdný	polotovar je v lisu 1
lis 1 je v dolní pozici	lis 1 je ve střední pozici

KA akcí, které řídí VB, jsou popsané podle jejich rozhraní. Rozhraní sestává z
- množiny rolí, které se zúčastní akce;
- externích objektů, které mohou být přístupné;
- předchozích a následných podmínek.

Předchozí a následné podmínky akcí jsou navržené, aby se dodrželo pořadí provedení akcí. Poté co KA akce bude provedená, předchozí podmínky další příslušné KA akce musí být splněné.

Aktivace další příslušné KA akce je hlavní úkol pro řadiče strojů (Vlákna).

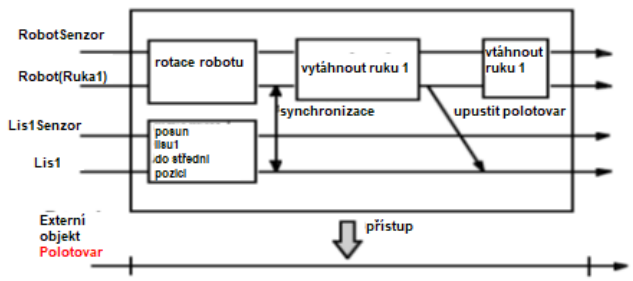
Aktivace KA akce se provádí jako vyvolání jejích rolí.

Polotovary

▪ Polotovary jsou předávané z jednoho stroje na další.

▪ Polotovar, který se nachází ve stroji, je prezentován v odpovídajícím řídicím procesu pomocí dvou proměnných:
- jedná proměnná (jméno)- pro identifikování polotovaru;
- druhá proměnná – pro vyjádření (označení) zdali polotovar již byl slisován. (na začátku polotovar není slisovaný).

Obrázek ilustruje interakcí mezi účastnícími Vlákny uvnitř akce *ZatíženíLisu1*



ZatíženíLisu1 akce má 4 role:
- Robot;
- Lis1;
- RobotSenzor;
- Lis1Senzor,

a reprezentuje kooperaci, která zařídí, že Ruka1 robota upustí polotovar do lisu 1.

1. Závady SW v Řídicím Programu

Kromě možné nekompletní nebo nekonzistentní specifikace výpočetních požadavků, existují další zdroje, které mohou vnést SW závady (bugs) do řídicího programu:

- První zdroj - je výběr neadekvátních nebo nedostatečných algoritmů, které nezahrnují všechny možné situace (zejména vzácné, neobyčejné).
- Druhý zdroj – jsou chyby při konvertování (převádění) vybraného algoritmu do programu. Zvlášť když se jedná o takové specifické prostředí jako VB.

- Navíc, závady SW mohou být kromě toho vnesené do té části programu, která koordinuje četné souběžné objekty (jelikož řídicí program zahrnuje souběžnost a distribuovanost).
- I když jednotlivé objekty mohou vyhovět specifikaci, specifikace pro kooperace mezi objekty je často nekompletní a nepřesná.

2. Zacházení se závadami SW v Řídicím Programu

Převažující závady SW, které mohly zůstat v řídicím programu, zahrnují:

1) Závady interakce (vzájemného působení).

Tyto závady vznikají když vzájemná součinnost mezi rolemi akce, mezi KA akcemi, mezi řídicím programem a VB nebyla správně specifikovaná nebo dostatečně analyzovaná.

2) Závady v časování.

Tyto závady vznikají když operace, role nebo KA akce není dokončená v předem definovaném termínu.

- Odhalení chyb a mechanismus obnovení jsou považované za část mechanismu podpory KA akce. Tento mechanismus je implementován jako znovu použitelný a dobře testovaný komponent.
- Mechanismus podpory KA akce rovněž obsahuje mechanismus řízení souběžnosti pro regulování přístupu do externích objektů. V VB-II externí objekty (tj. kovové polotovary) nemohou být sdílené dvěma a více souběžnými KA akcemi v zájmu bezpečnosti.

Navzdory veškerým úsilím odstranit závady SW se závady SW mohou nacházet v Řídicím Programu!

	Přechodné závady SW	Závady interakcí	Závady v časování
Projev	Vzniká v řídicím programu když ruka1 robota musí umístit polotovar do neobsazeného lisu	Vzniká pouze, když více než dva polotovary jsou dodané do VB	Role KAA nebo sama KAA není dokončená v předem definovaném termínu
Zacházení se závadami	"Re-try" strategie (opakování)	Rozmanitost návrhu	Timeout mechanismus (opakování nebo abortování)

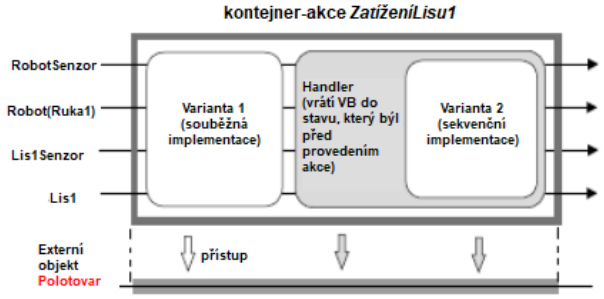
Rozmanitost návrhu

- Náhodná rozmanitost může být dosažená různými programátory a návrháři. Avšak záměrně vybrané rozmanité struktury dat a algoritmy jsou méně pravděpodobně selžou současně. Takový postup se jmenuje "nucená rozmanitost".

- Pro VB-II (pro KAA – *ZatíženíLisu1*) dvě SW varianty byly navržené s použitím nucené rozmanitosti:
  - První Varianta zahrnuje několik souběžných aktivit a dvě souběžně vnořené akce *RotaceRobotu* a *PosunLisu1DoStředníPozice*
  - Druhá Varianta je navržená tak aby poskytla stejnou funkčnost ale na základě jednoduššího algoritmu bez jakékoli souběžnosti. Uvnitř druhé varianty všechny vnořené akce jsou prováděné postupně, v následujícím pořadí: *PosunLisu1DoStředníPozice, RotaceRobotu, VytáhnoutRuku1, VtáhnoutRuku1.*

- Protože interakce mezi těmito vnořenými KA akcemi jsou podstatně odlišné v různých variantách, pravděpodobnost, že varianty selžou stejně, by měla být malá.

Obrázek ilustruje strukturu KA akce *ZatíženíLisu1*. Akce navržená s použitím dynamické nadbytečnosti pro zajištění odolnosti proti SW závadám.



KA akce *ZatíženíLisu1* je navržená jako kontejner-akce. Kontejner se skládá ze dvou rozmanitě navržených variant. Normálně, kontejner-akce vykonává pouze první variantu. Jestli chyba nebude odhalená, kontejner-akce skončí a vytváří normální výsledek. Ale jestli chyba bude odhalená (buď pomocí Tvrzení nebo PT), tato chyba musí být signalizovaná do kontejner-akce.

- Když vznikne chyba ve Variantě 1, bude signalizována odpovídající výjimka do kontejneru.
- Pak bude vyvolán odpovídající Handler, který musí obnovit stavy objektů zapojených do akce strojů a polotovaru. Handler vrátí stavy objektů a polotovaru do původního stavu (tj. do stavu, který byl před provedením akce *zatíženíLisu1).*
- Po ukončení obnovení, Handler vyvolává druhou Variantu v naději, že stejná chyba nebude opakována.
- V nejhorším případě, když chyba (možná není stejná) vznikne opět, existuje možnost pro akci *ZatíženíLisu1* signalizovat výjimku „Abort" (jestli obnovení stavů bylo úspěšné) nebo výjimku „Selhání" do obalující akce.

Příklad. Chyba při pozicování Ruky1 Robota.

Handler musí ošetřit výjimečnou situaci, když vnořená akce *VtáhnoutRuku1* selže přivést Ruku do správné pozice. Handler může použit pro akce jak strategie opakování tak i rozmanitost návrhu variant.

Základní funkčnost Handleru. Handler pro ošetření chyby „Pozice Ruky1":

Handler opakuje akce *VtáhnoutRuku1* a kontroluje pozice Ruky1. Jestliže stejná chyba bude setrvat při každém opakování akce (počet opakovaní je předem určený), to Handler obnoví stavy polotovaru a všech strojů zapojených do akce (tj. Robot a Lis1). Dále Handler vyvolá druhu variantu. Jestli varianta bude signalizovat výjimku místo Normálního výsledku, Handler znovu obnoví stavy. Jestli obnovení proběhně úspěšně, Handler signalizuje výjimku „Abort" do kontejner-akce. Jinak Handler signalizuje výjimku „Selhání".

Literatura:

1.C.Lewerentz and T. Lindner. *Formal Development of Reactive systems: Case study. "Production Cell",* LNCS-891, Springer, Jan. 1995.

2. A. Lötzbeyer. Task Description of a Fault-Tolerant Production Cell. Version 1.6, available from http://www.fzi.de/prost/projects/korsys/korsys.html , 1996.

3. J.Xu, A.Romanovsky, A.Zorzo, B.Randell, R.J.Stroud and E.Canver. Developing Control Software for Production Cell 2: Failure Analysis and System Design Using CA Actions. 3rd Year Report, *ESPRIT Long Term Research Project 20072 on Design for Validation,* Nov. 1998.

Kontrolní otázky:

- specifikujte bezpečnostní požadavky na řídící program VB
- definujte selhání VB
- vysvětlete způsoby odhalení selhání VB
- popište návrh řídícího programu s použitím KAA

Úkoly pro samostatnou práci:
Prostudovat případovou studia. Použit prezentaci ve výkladové části.

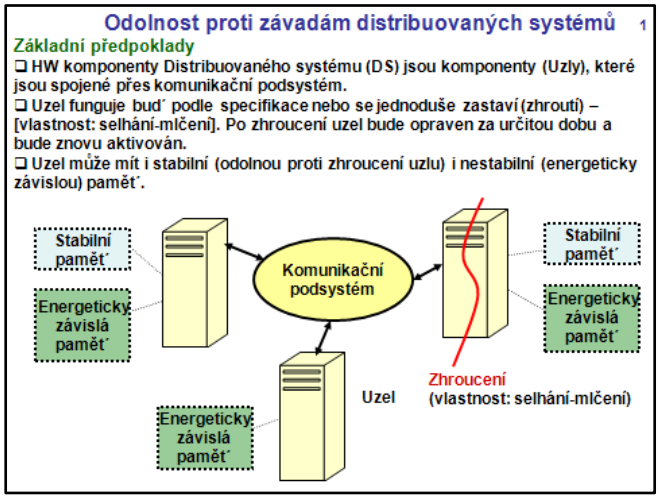
Kapitola 10. Dependabilita distribuovaných aplikací

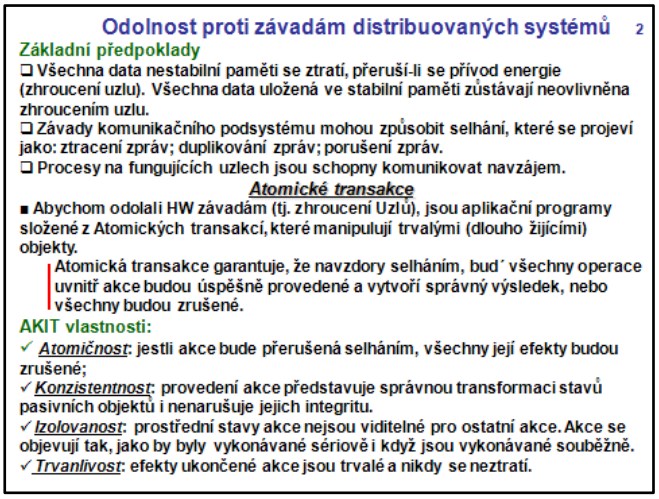
Cíl kapitoly:

- vysvětlit jak může být docílena dependabilita distribuovaných aplikací
- probrat hlavní moduly architektury distribuovaných aplikací

Klíčová slova: průhlednostní vlastnosti, modul distribuovaného systému, atomická akce, dálkové volání procedury, trvalý objekt

Výkladová část:
Následující prezentace vysvětluje hlavní cíly kapitoly





Odolnost proti závadám distribuovaných systémů 3

Atomické transakce

- Transakce může být ukončena dvěma způsoby:
  - committed;(všechny operace jsou vykonané správně a jejich efekty uložené ve stabilní paměti)
  - abortovaná (odrolování).

Vlastnosti atomických transakcí

1) Serializovanost;
2) Atomičnost selhání;
3) Trvanlivost efektu.

Serializovanost – zajišťuje, že současná vyvolání sdílených objektů se nebudou ovlivňovat navzájem (tj. kterékoli současné provedení může být představeno jako posloupnost jednotlivých provedení).

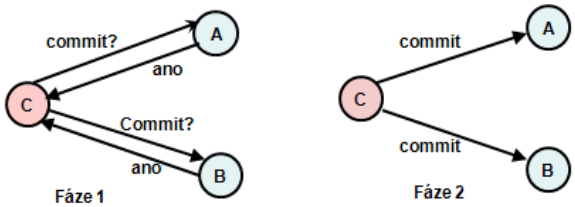
Atomičnost selhání – zajišťuje, že akce bude ukončená buď normálně (committed) a vytvoří očekávané správné výsledky anebo bude abortovaná, což nevytvoří žádné výsledky a neprovede žádné změny ve stavech objektů.
- Typickými selháními, která vyžadují abortování, jsou zhroucení uzlu a komunikační selhání (např. trvalé ztracení zpráv).

Trvanlivost efektu – kterékoli committed změny stavů (tj. nové stavy objektů) byly uložené do stabilní paměti.

Odolnost proti závadám distribuovaných systémů 4

Dvoufázový commit protokol

- Dvoufázový Commit protokol garantuje, že všichni účastníci akce buď commit nebo abortují všechny změny provedené v průběhu transakce.



Fáze 1: Koordinátor akce, C, se snaží komunikovat se všemi účastniky akce, A a B, aby zjistil jestli účastníci hodlají provést commit nebo abort (odpověď Abort kteréhokoli z účastníků vystupuje jako veto, což vede k tomu, že celá akce bude abortována).
Fáze 2: Koordinátor nutí účastníky provést svoje rozhodnutí (commit nebo abort).

- Po odpovědi na dotaz koordinátora (ve fázi 1) každý účastník, který vrátil odpověď Ano, musí zůstat zablokovaný, dokud nedostane zprávu (příkaz) od koordinátora ve fázi 2.

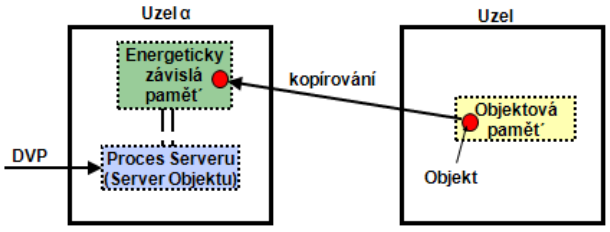
Odolnost proti závadám distribuovaných systémů 5

Průhlednostní vlastnosti:

- **Distribuovaný systém musí podporovat některé požadované „průhlednostní" vlastnosti, což znamená, že distribuovaný systém (když budeme potřebovat) se chová jako jeho nedistribuovaný protějšek.**

❑ Průhlednost přístupu zajišťuje jednotný způsob vyvolání operací jak lokálních tak i vzdálených objektů. Všechny síťové komunikace jsou skryté před volajícím.
❑ Průhlednost umístění. Nepotřebujeme vědět, kde je objekt umístěn. K přístupu do objektu postačí pouze jeho Jméno.
❑ Průhlednost migrace. Objekty mohou být přemístěny z jednoho uzlu do druhého (aby se zlepšila odolnost proti závadám i/nebo výkon). Přemístění objektu není viditelné pro uživatele (klienty) objektů.
❑ Průhlednost souběžnosti zajišťuje neovlivněný přístup ke sdíleným objektům v přítomnosti souběžných vyvolání.
❑ Průhlednost replikace. Pro zvýšení dostupnosti objektů (a pro zajištění odolnosti proti HW závadám) můžeme použít několik replik objektu. Složitost mechanismu, který zajišťuje konzistentnost replik je ukrytá před uživatelem.
❑ Průhlednost selhání. Může být dosažená použitím nadbytečnosti pro maskování selhání a prostředků pro obnovení systému.

Odolnost proti závadám distribuovaných systémů 6



- Trvalý objekt se normálně nachází v pasivním stavu. Tento stav je uložen v objektové paměti nebo v objektové DB a může být aktivován požadavkem (tj. když bude vyvolán).
- Aktivace objektu znamená zavádění jeho stavu a metod z objektové paměti do nestabilní paměti a přidružení serveru pro přijímání DVP (dálkového volání procedury).
  - ▽ Normálně trvalý stav objektu je umístěn v objektové paměti na jednom uzlu. Nicméně dostupnost objektu může být zvýšená pomocí replikace objektu (tj. umístěním objektu ve více než jedné objektové paměti).
- Repliky objektu mohou být řízené pomocí příslušných protokolů zajišťujících konzistentnost replik.

Odolnost proti závadám distribuovaných systémů 7

Model Klient / Server pro přístup a manipulaci trvalými objekty

- Předpokládá se, že pro každý Trvalý Objekt existuje alespoň jeden Uzel α, který když je v provozu, je schopen spustit (vykonávat) Server objektu. Server bude vykonávat operace objektu.
- Před tím než Klient vyvolá operace na objektu, musí být objekt nejprve připojen nebo vázán do Serveru, který bude řídit objekt.
- Pokud se požadovaný objekt nachází v pasivním stavu, je Uzel α rovněž zodpovědný za aktivaci objektu dříve než připojí Klienta do Serveru.

Aby se provedlo připojení, Aplikace (program) musí být schopná získat (obdržet) informace o umístění objektu (jako např. Jméno Uzlu, na kterém Server pro objekt může být dostupný).

Získání informace o umístění objektu

Krok 1: Aplikace uvádí (prezentuje) jméno objektu (řetězec) globálně dostupné Jmenující Službě. Tato služba mapuje tento řetězec na UID objektu.
Krok 2: Aplikace uvádí UID objektu globálně dostupné Vázající (binging) Službě, aby získala informací o umístění objektu.

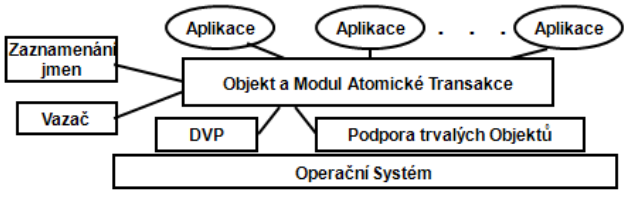
- Jakmile Aplikace získá informaci o umístění objektu, může požádat příslušný Uzel o připojení k serveru, který bude řídit tento objekt.

Odolnost proti závadám distribuovaných systémů 8

Hlavní Moduly Distribuovaného Systému:

➢ ***Modul Atomické Akce***: poskytuje podporu Atomické akce pro Aplikace ve formě operací: start, commit a abort Atomické akce.
➢ ***Modul DVP***: poskytuje klientům prostředky pro připojení (odpojení) k Serverům objektů a vyvolání operací na objektech.
➢ ***Modul zaznamenání jmen (Jmenující modul)***: zajišťuje mapování jmen objektů, stanovených uživatelem, na UID (unikátní ID).
➢ ***Modul vazby***: zajišťuje mapování UID na informace o umístění (jako např. Identita Uzlu, na kterém může být dostupen server pro objekt).
➢ ***Modul podpory trvalých objektů***: poskytuje objektům servery a zajišťuje přístup objektů ke stabilní paměti.

Vztah mezi moduly



Odolnost proti závadám distribuovaných systémů 9

■ Aplikace přímo užívá služby, které poskytuje Modul Atomické Akce. Tento Modul je zodpovědný za řízení přístupu k ostatním modulům.

Modul Atomické Akce

Příklad jednoduchého Programu

```
1. Příklad P ("tento"); // připojení do serveru
2. AtomickáAkce A;
3. A.Begin(); // start atomické akce A
4. P.op(); // vyvolání operací „op“ na objektu P
5. if (...) A.Abort(); // abortování atomické akce A
6. else A.End(); // commit atomické akce A
```

■ Aplikace (program) modifikuje vzdálený trvalý Objekt, který se jmenuje *Tento*. Objekt třídy Příklad má možnost vrátit svůj původní stav v případě, že některá podmínka nebude splněná.

■ Aplikace vytvoří konkretizaci (instanci) Atomické akce, nazvanou A. Pak začne Akce, která provádí operace na objektu. Akce bude Commit nebo Abortována.

∇ Předpokládá se, že tento Program nejprve bude zpracován Stub Generátorem, který je jazykově orientovaný (tj. specifický pro každý jazyk). Stab Generátor má za cíl zpracovat Program uživatele a vygenerovat Klient/Server kód potřebný pro přístup ke vzdáleným objektům pomocí DVP.

Odolnost proti závadám distribuovaných systémů 10

Modul Dálkového Volání Procedury (DVP)

■ Modul DVP opatřuje odlišné rozhraní Klienta a Serveru :

- iniciovat
- ukončit
- vyvolání

} Rozhraní Klienta

get_request
send_reply

} Rozhraní Serveru

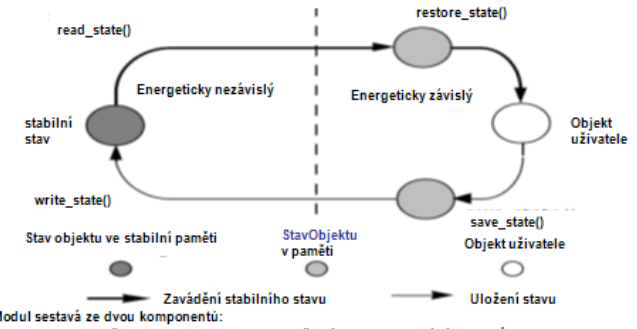
■ Operace stanovené modulem DVP nejsou obecně přímo používané Aplikací. Tyto operace jsou používané vygenerovanými Stuby (pro Klienta a Server).

■ Klienty a Servery mají komunikační identifikátory, KIDy, pro odesílání a přijímání zpráv.

∇ Modul každého Uzlu má ***manažera spojení***, který je zodpovědný za vytvoření a zrušení spojení s Lokálními servery.

Odolnost proti závadám distribuovaných systémů 11

Modul podpory trvalých objektů



■ Modul sestává ze dvou komponentů:

1) Objekt-manažer komponent. Je zodpovědný za poskytování serverů pro objekty (tj. za udržování mapování mezi UID aktivovaných objektů a odpovídajícími servery).

2) Komponent pro ukládání objektů. Vystupuje jako vstup do podsystému Lokální objektové paměti. Ukládá a vyzvedá instances (konkretizaci) třídy *StavObjektu* (operace: read_state and write_state)

∇ Instance StavObjektu je nezávislá (na Uzlu) prezentace stavu pasivního objektu, která je vhodná pro přenos mezi Uzly.

Odolnost proti závadám distribuovaných systémů 12

Modul záznamu jmen (Jmenující modul) a Modul vazby

Jmenující modul: Lookup (vyhledat) → mapuje Jméno Objektu (řetězec) na UID

Modul Vázání Locate (lokalizovat) → mapuje UID na umístění (JménoUzlu)

Zajištění průhlednosti Migrace a Replikace

Migrace

Jednoduchý způsob: dovolit aktivaci Objectu nejenom v Uzlu, ve kterém je Objekt uchován.

■ Uzly bez objektové paměti mohou spustit Server objektu; takový Uzel bude obsahovat Modul podpory trvalých objektů, ale bez jeho komponentu pro ukládání objektů.

∇ Předpokládá se, že Manažer objektu (komponent modulu podpory trvalých objektů) ted´ už nepodporuje mapování (spojení) mezi UID aktivovaných objektů a Servery. Tato informace je záležitost Služby Vázání.

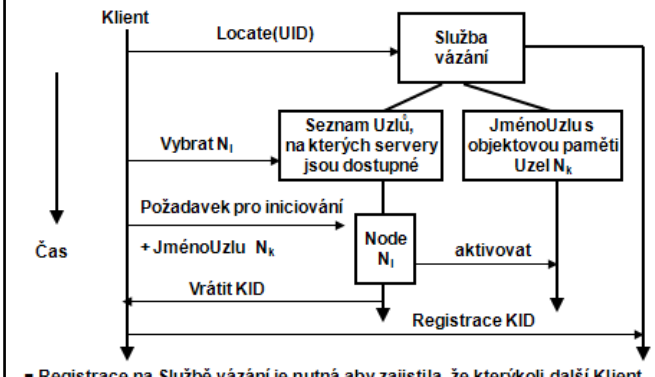
■ Pro Pasivní objekt, funkce Locate(UID) služby vázání vrátí klientu JménoUzlu, na kterém se nachází Objektová pamět´ a také vrátí seznam uzlů, na kterých mohou být dostupné Servery pro tento Objekt.

■ Pro Aktivní objekt, pár (JménoUzlu, KID) ukazuje KID serveru objektu, který zpracovává Objekt na Uzlu (s odpovídajícím jménem = “JménoUzlu”).

Odolnost proti závadám distribuovaných systémů 13

Migrace

Aktivace pasivního objektu.



■ Registrace na Službě vázání je nutná aby zajistila, že kterýkoli další Klient bude spojen s týmž Serverem.

Kontrolní otázky:

- charakterizujte základní předpoklady pro distribuovaný systém
- popište dvoufázový commit protokol
- charakterizujte hlavní moduly distribuovaného systému

Úkoly pro samostatnou práci:
Prostudovat odolnost distribuovaných systémů proti závadám. Použit prezentaci ve výkladové části.

Kapitola 11. Použití skupin objektů pro zajištění odolnosti proti závadám

Cíl kapitoly:

- vysvětlit použití replikace objektu pro zajištění odolnosti proti závadám
- probrat tři aspekty řízení konzistenci replik
- vysvětlit replikační politiky

Klíčová slova: replika, řízení konzistenci replik, aktivace objektu, replikační politiky

Výkladová část:
Následující prezentace vysvětluje hlavní cíly kapitoly

Odolnost proti závadám distribuovaných systémů 14

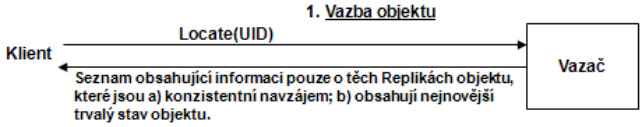
Replikace

- Dostupnost Objektu může být zvýšena pomocí Replikace objektu na několika Uzlech, což znamená, že stav objektu bude uchován ve více než jedné Objektové paměti.
  - ∇ Repliky Objektu musí být řízené přes vhodný protokol zajišťující konzistentnost replik. Repliky musí být konzistentní navzájem.

Silná konzistentnost == Trvalé stavy všech replik musí být identické.

Tři aspekty řízení konzistencí replik:
1. Vazba Objektu.
2. Aktivace Objektu a přístup k Objektu.
3. „Commit" zpracování.

1. Vazba objektu



- Jestli Objekt už je aktivován, Vazač dovoluje připojení ke všem Serverům, které zpracovávají Repliky aktivovaného Objektu.

Odolnost proti závadám distribuovaných systémů 15

Replikace

2. Aktivace objektu a přístup k objektu

- Pasivní objekt musí být aktivován podle stanovené Replikační Politiky:
  - aktivní replikace;
  - koordinátor- skupina (kohorta) pasivní replikace;
  - jedna kopie pasivní replikace.

Aktivní replikace- více než jedna kopie pasivního objektu jsou aktivované na různých Uzlech a všechny aktivované kopie budou zpracované.
Koordinátor-kohorta pasivní replikace- několik kopií Objektu je aktivováno. Nicméně pouze jedna replika (koordinátor) bude zpracována. Koordinátor pravidelně „checkpoints" (předává) svůj stav ostatním replikám (tj. kohortě).
Jedna kopie pasivní replikace- pouze jedna kopie je aktivována. Aktivovaná kopie pravidelně ukládá (checkpoints) svůj stav do objektových pamětí, ve kterých jsou stavy uchovávané.

3. „Commit" zpracování

- Jakmile Aplikace ukončí používání Objektu, je nutné aby nové stavy vzájemně konzistentních replik objektu byly zaznamenané (uložené) v objektových pamětech. Uložení stavů replik objektu probíhá v době provedení operace „Commit" Atomické akce (Aplikace).

Příklad: Aplikace modifikuje Objekt A. Používá se politika aktivní replikace.
- Při startu Aplikace jsou dostupné dvě repliky objektu (A_1 a A_2), Předpokládá se, že uzel obsahující repliku A_1 ,se zhroutil. Takže pouze replika A_2 bude modifikována.
- V době provedení operace „Commit" informace o objektu A, která je udržována Službou vázání, musí být modifikována, aby se vyloučilo A_1 ze seznamu dostupných replik objektu A.

Odolnost proti závadám distribuovaných systémů 16

Řízení replikami

- Jmenující služba a služba vázání udržují dvě množiny dat, které se týkají Uzlů:

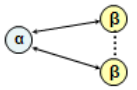
i. Pro Objekt A, množina S_{TA} obsahuje jména těch uzlů, jejichž objektové paměti umíst´ují stavy objektu A;
ii. Množina S_{VA} obsahuje jména uzlů, které jsou schopny spustit server pro zpracování objektu A.

Objekt se stává nedostupným, když se všechny Uzly∈ S_{VA} zhroutí a/nebo se všechny Uzly∈ S_{TA} zhroutí.

1) $|S_{VA}| = |S_{TA}| = 1$ - případ nereplikovaného Objektu

Jestliže se buď α nebo β zhroutí v průběhu provedení Atomické akce, akce musí být abortována.

2) $|S_{VA}| = 1$ & $|S_{TA}| > 1$ - případ, když pouze stav objektu je replikován.



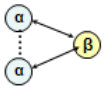
Atomická akce, která používá objekt A, musí být abortována, když se Uzel α zhroutí nebo se všechny Uzly∈ S_{TA} zhroutí

Jedna kopie pasivní replikace

Odolnost proti závadám distribuovaných systémů 17

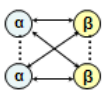
Řízení replikami

3) $|S_{VA}| > 1$ & $|S_{TA}| = 1$ - případ, když Servery jsou replikované, ale pouze jeden Stav objektu je dostupný.



Toto schéma může odolat do (k-1) selhání Serverů

4) $|S_{VA}| > 1$ & $|S_{TA}| > 1$ – všeobecný případ



Každý Server může volně „rozhodnout" ze kterého Uzlu∈ S_{TA} zavést stav objektu.

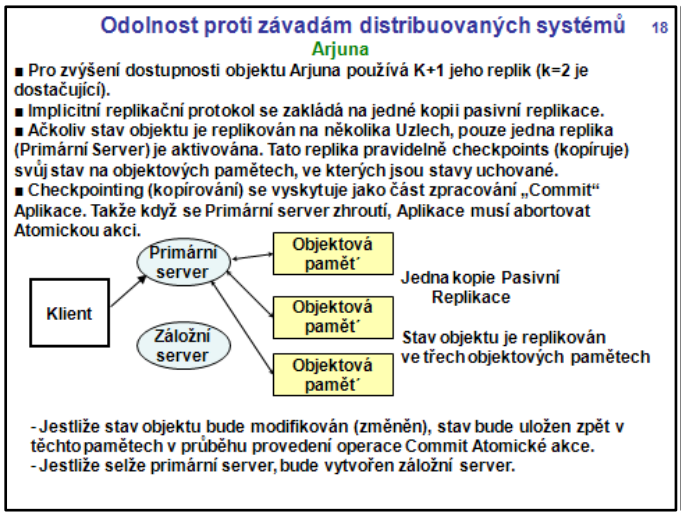
Arjuna [1,2,3,4]

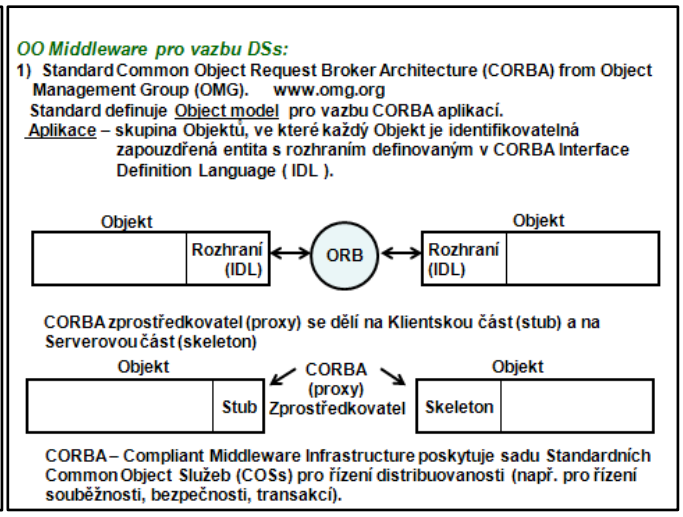
Arjuna je OO programovací systém, implementovaný v C++, který poskytuje množinu nástrojů pro návrh distribuovaných Aplikací odolných proti závadám.

- Arjuna používá Atomické akce (transakce) a Replikace.

Replikace objektů v Arjuna

Arjuna implementuje Silnou Konzistentnost (stavy všech replik jsou identické).





Kontrolní otázky:

- v čem spočívá řízení konzistenci replik
- definujte a popište replikační politiky
- vysvětlete, jak probíhá aktivace objektu

Úkoly pro samostatnou práci:

Prostudovat použití skupin objektů pro zajištění odolnosti proti závadám. Použit prezentaci ve výkladové části.

Kapitola 12. Metoda Event-B. Formální verifikace.

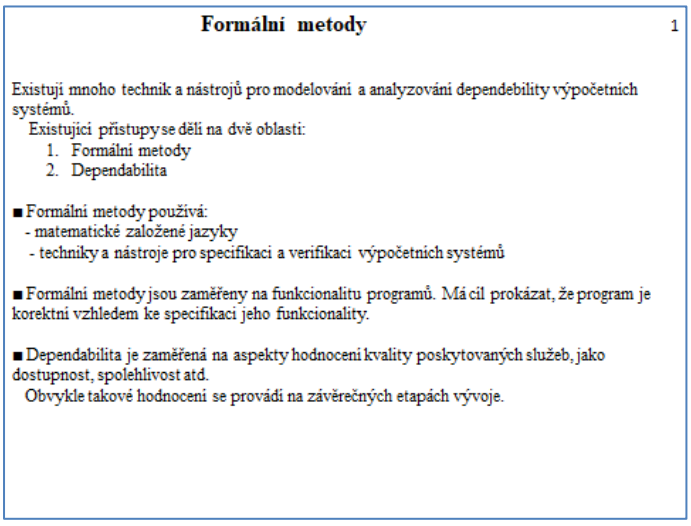
Cíl kapitoly:

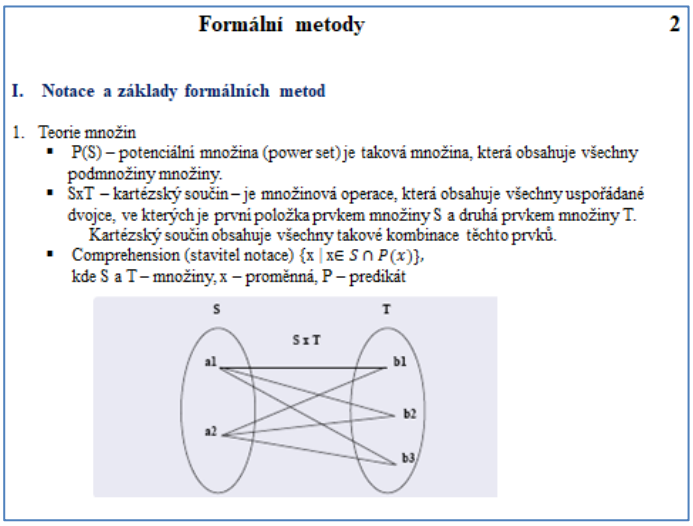
- charakterizovat formální metody
- probrat notace a základy formálních metod
- probrat notaci Event-B modelování
- vysvětlit průkazové povinnosti

Klíčová slova: teorie množin, machines (automaty) a contexts (kontexty), Events (události), Průkazové povinnosti

Výkladová část:

Následující prezentace pomůže studentům splnit úkoly samostatné práce.





Formální metody 3

I. Notace a základy formálních metod

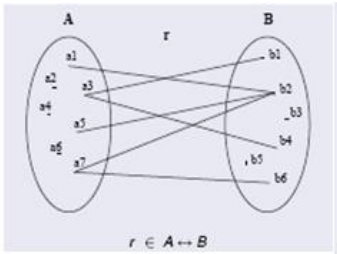
1. Teorie množin
   Elementární operace s množinami

Union	S ∪ T
Intersection (průnik)	S ∩ *T*
difference	S \ T
extension	{a,.., b}
Empty set (prazdná)	Ø
	S x T
	P(S)
	{x | x∈ *S* ∩ *P*}
Set inclusion	S⊆ *T*
	S=T

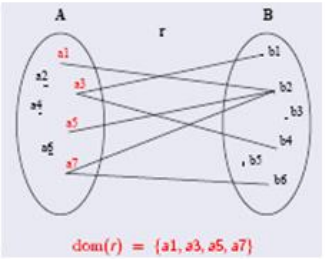
Formální metody 4

1. Teorie množin
   Binární vztahové operátory

Binární vztahy	S ⟷ T
domain	dom (r)
range	ran (r)
converse	r^{-1}



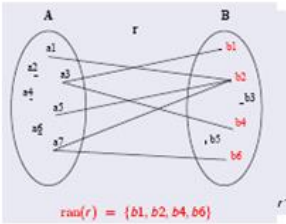
$r \in A \leftrightarrow B$



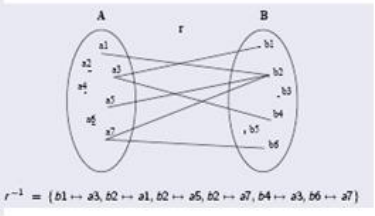
$dom(r) = \{a1, a3, a5, a7\}$

Formální metody 5

1. Teorie množin
   Binární vztahové operátory



$ran(r) = \{b1, b2, b4, b6\}$

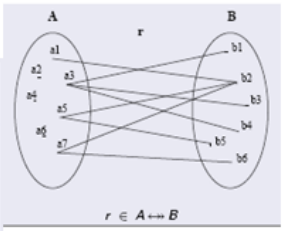


$r^{-1} = \{b1 \mapsto a3, b2 \mapsto a1, b2 \mapsto a5, b2 \mapsto a7, b4 \mapsto a3, b6 \mapsto a7\}$

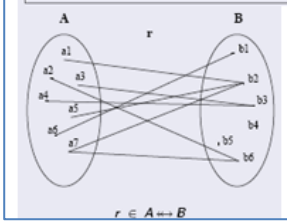
Formální metody 6

1. Teorie množin
Binární vztahové operátory členství (memberships)

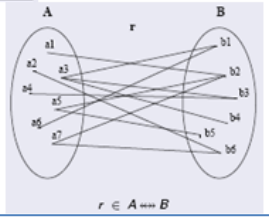
Partial surjective binary relations	$S \leftrightarrow\!\!\!\!\!\rightarrow T$
Total binary relations	$S \leftrightarrow\!\!\!\!\!\!\rightarrow T$
Total surjective binary relations	$S \leftrightarrow\!\!\!\!\!\!\!\rightarrow T$



$r \in A \leftrightarrow B$



$r \in A \leftrightarrow\!\!\!\!\!\rightarrow B$

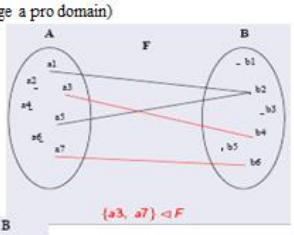


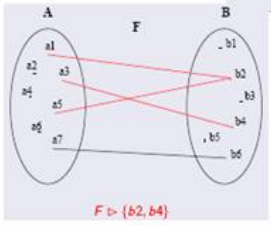
$r \in A \leftrightarrow\!\!\!\!\!\!\rightarrow B$

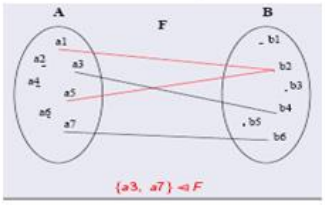
1. Teorie množin

Operátory omezení (pro range a pro domain)

Domain restriction	$S \lhd r$
Range restriction	$r \rhd T$
Domain subtraction	$S \lhd\!\!\!- r$
Range subtraction	$r \rhd\!\!\!- T$





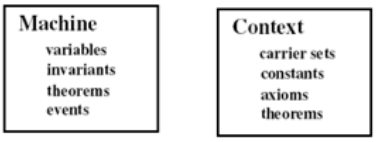


II. **Notace Event-B modelování**

A. Machines (automaty) a Contexts (kontexty)

- Primárním pojmem při formálním vývoji pomocí Event-B je model
- Model obsahuje kompletní matematický vývoj systému diskrétních přechodů
- Tento systém je tvořen několika komponentami dvou typů : mechines a contexts

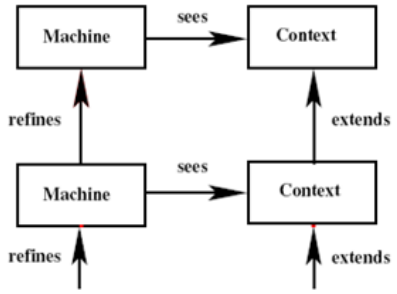
■ Machines (automaty) obsahují: variables, invariants, theorems a events
■ contexts (kontexty) obsahují: sets, axioms a theorems



■ Machines a kontexty maji rozmanité vztahy:
- Machine může být upřesněn jiným machine (automatem)
- Kontext může být rozšířen jiným kontextem (cykly nejsou povoleny)
- Machine může vidět jeden nebo několik kontextů

II. **Notace Event-B modelování**

A. Machines (automaty) a Contexts (kontexty)



A. Machines (automaty) a Contexts (kontexty)

machine
$< machine\_identifier >$
refines ⋆
$< machine\_identifier >$
sees ⋆
$< context\_identifier >$
...
variables
$< variable\_identifier >$
...
invariants
$< label >: < predicate >$
...
theorems ⋆
$< label >: < predicate >$
...
events
$< event >$
...
variant ⋆
$< variant >$
end

context
$< context\_identifier >$
extends ⋆
$< context\_identifier >$
...
sets ⋆
$< set\_identifier >$
...
constants ⋆
$< constant\_identifier >$
...
axioms ⋆
$< label >: < predicate >$
...
theorems ⋆
$< label >: < predicate >$
...
end

Events (události)

Event reprezentuje přechod z jednoho stavu do druhého
■ event zahrnuje guards (stráže) a actions (akce)
■ guards označují nutné podmínky pro událost aby se mohla uskutečnit
■ akce reprezentuji způsob, jak se změní (modifikuji se) proměnné
■ event může být jednoduchým a guarded (klíčové slovo „where“) nebo může být parameterized a guarded (klíčové slovo any i where)
■ kdy event je umistěn uvnitř machine, který upřesňuje jiný machine, to tento event může specifikovat abstraktni event(s). Klíčové slovo „refines“.
■ kdy upřesňujici event upřesňuje abstraktni event, který je parametrizován, je možno zajistit „some witnesses“ (klíčové slovo with).

Guards, witnesses a actions máji jména (tj. are labelled).
■ *Status*:
- obvykle je „normal“
- convergent je nový event v refined machine, který nebyl v abstraktni machine. Všechny convergent events se týkaji sekci Variant.
- anticipated event je nový event, který ještě neni convergent ale by se měl stát v následném refinement (upřesnění).

■ když event je „normal“, znamená to, že tento event se netýká sekci Variant

Events (události)

$< event\_identifier > \widehat{=}$
status
$\{normal, convergent, anticipated\}$
refines ⋆
$< event\_identifier >$
...
any ⋆
$< parameter\_identifier >$
...
where ⋆
$< label >: < predicate >$
...
with ⋆
$< label >: < witness >$
...
then ⋆
$< label >: < action >$
...
end

■ *Variant*
Sekce Variant se vyskytuje v refined machine, který obsahuje některé „convergent“ events.
Tato sekce obsahuje buď přirozené číslo (natural), které musi být snižováno každým „convergent „ event.

■ *Actions*
Akce mohou být deterministické. V daném případě akce se skládá z odlišných identifikátorů proměnných následovanými značkou := A následovanými seznamem výrazů, tj.

<variable_identifier_list>:=<expression_list>

Events (události)

Příklad.

x, y := x + z, y − x

Kde x a y – jsou modifikovány. Zatímco proměnná z není modifikována.

■ Alternativně, akce může být nedeterministická. V daném případě akce je tvořená jako
<variable_identifier_list> :| <before_after_predicate>

■ Before_after_predicate může obsahovat vše proměnné machine, které označují odpovídající hodnoty těsně před akcí.
Tento predikát může také obsahovat některé jména proměnných ze seznamu, které označují odpovídající hodnoty těsně po provedení akce.

Příklad.
x, y :| x' >y ^ y'>x' +z
Proměnná x stává větší než y. Proměnná y stává větší než x' (nová hodnota pro x).
Proměnná z se nemodifikuje.

■ Další možnost je definovat nedeterministickou akce jako
<variable_identifier> : ∈ < *set_expression* >
Tento zápis je zvláštní případ předchozího zápisu.

Příklad. Machine má proměnné A, x a y.
x: ∈ A U {y} nebo x: | x' ∈ A U {y}
Proměnná x stává prvkem množiny A U {y}, kde proměnné A a y se nemodifikují.

Events (události)

■ Různé akce ve stejném event se nemohou zabývat ze stejnými proměnnými.
Tohle není možné
x:=5 první akce
x: | x' > x druhá akce

■ ***Witnesses*** (svědectví)
Kdy konkrétní event refines (upřesňuje) abstraktní event, který je parametrizován, to všechny abstraktní parametry musí dostat hodnotu v konkrétním event.
Witness je definován pomocí predikátu zahrnujícího abstraktní parametr.
Většinou taký predikát je jednoduchá rovnost.

Events (události)

■ ***Witnesses*** (svědectví)
Následující příklad ukázuje dvě witnesses

pass $\widehat{=}$
any
p
l
where
$grd1: \ p \mapsto l \in aut$
$grd2: \ sit(p) \mapsto l \in com$
then
$act1: \ sit(p) := l$
end

new_pass $\widehat{=}$
refines
pass
any
d
where
$grd1: \ d \in ran(dap)$
with
$p: \ p = dap^{-1}(d)$
$l: \ l = dst(d)$
then
$act1: \ sit(dap^{-1}(d)) := dst(d)$
$act2: \ dap := dap \rhd \{d\}$
end

Když konkrétní event je taky parametrizován, to abstraktní parametr, který je tentýž jako konkrétní, nepotřebuje explicitního witness.

- ❑ Konečním cílem formální metody je prokázat vlastnosti specifikace.
- ❑ Vlastnosti specifikace jsou uvedeny v sekce **INVARIANT**
- ❑ Nástrojová podpora generuje ***důkazové povinnosti***, které musí být odváděny, aby ověřil, že invarianta (tj. vlastnost specifikace) je udržována.

Priority operátorů v Event-B (from low to higher):

$\forall x \cdot P$ and $\exists x \cdot P$ (mixing allowed)

$P \Rightarrow Q$ and $P \Leftrightarrow Q$ (mixing not allowed)

$P \wedge Q$ and $P \vee Q$ (mixing not allowed)

$\neg P$	$\Leftrightarrow$	U+21D4	LOGICAL EQUIVALENCE
	$\Rightarrow$	U+21D2	LOGICAL IMPLICATION
	$\wedge$	U+2227	LOGICAL AND
	$\vee$	U+2228	LOGICAL OR
	$\neg$	U+00AC	NOT SIGN
	$\top$	U+22A4	TRUE PREDICATE
	$\bot$	U+22A5	FALSE PREDICATE
	$\forall$	U+2200	FOR ALL
	$\exists$	U+2203	THERE EXISTS

Příklady (jak tyto priority jsou používány aby odstranil závorky):

$P \wedge Q \Rightarrow R$	is parsed as	$(P \wedge Q) \Rightarrow R$
$\forall x \cdot \exists y \cdot P$	is parsed as	$\forall x \cdot (\exists y \cdot P)$
$\forall x \cdot P \Rightarrow Q$	is parsed as	$\forall x \cdot (P \Rightarrow Q)$
$\forall x \cdot P \wedge Q$	is parsed as	$\forall x \cdot (P \wedge Q)$
$\forall x \cdot \neg P$	is parsed as	$\forall x \cdot (\neg P)$
$\neg P \Rightarrow Q$	is parsed as	$(\neg P) \Rightarrow Q$
$\neg P \wedge Q$	is parsed as	$(\neg P) \wedge Q$

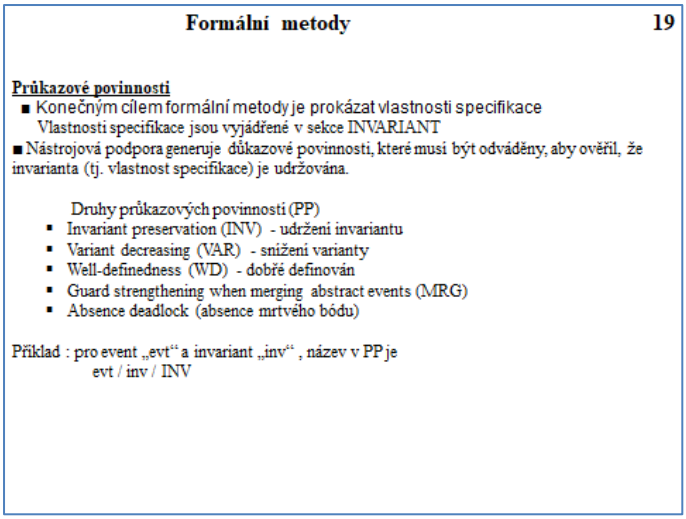
Notace
- svislitko | označuje změnu (alternation)
- hranaté závorky [ ] zahrnuji volitelnou část (optional part)

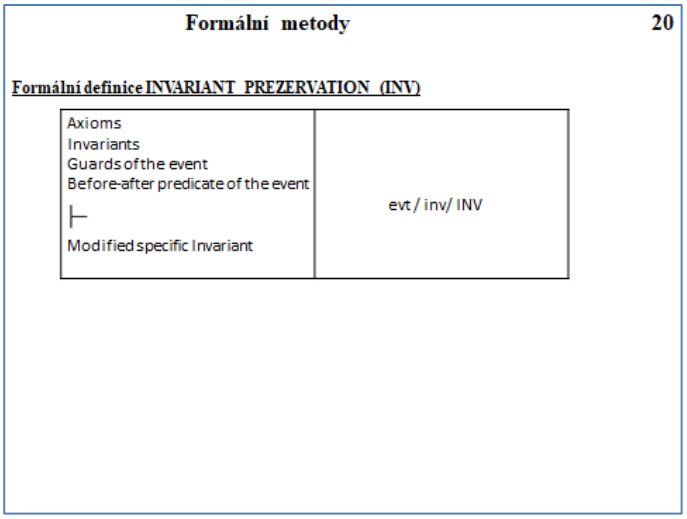
Výrazy
- dvojice výrazů jsou tvořeny pomocí maplet operátoru: $\mapsto$
- čárka je vždy oddělovač
- maplet je konstruktor dvojic

Příklady:

Classical-B	Event-B
$x, y \in S$	$x \mapsto y \in S$
$x, y = z, t$	$x \mapsto y = z \mapsto t$
$f(x, y)$	$f(x \mapsto y)$

Set comprehension
Jsou dvě formy set comprehension:
1. Nejvíce obecná je
{x. P(x) | E(x) }
popisuje množinu, jejichž prvky jsou E(x) . Pro všechny x, pro které P(x) je platná.
Např. množina všech sudých přirozených čísel může být zapsána jako
{x . x ∈ *N* | 2 × x}
2. Druhá forma je kratší a je více kompaktnější: {E | P }.
zde proměnná neuvedena explicitně. Pro náš příklad máme {2 × x | x ∈ N }





Kontrolní otázky:

- na co zaměřeny a jaký mají cil formální metody
- definujte a popište hlavní komponenty modelu v Event-B
- co reprezentují Events (udalosti) v Machines (automatech)

Úkoly pro samostatnou práci:
1) Prostudovat teorie množin. Použit obsah výkladové části.
2) Navrhnout jednoduchý model (Event-B): Machines (automaty) a Contexts (kontexty). Použit literaturu [J.R. Abrial. Modeling in Event-B: System and Software Engineering. Cambridge University Press, 2010].

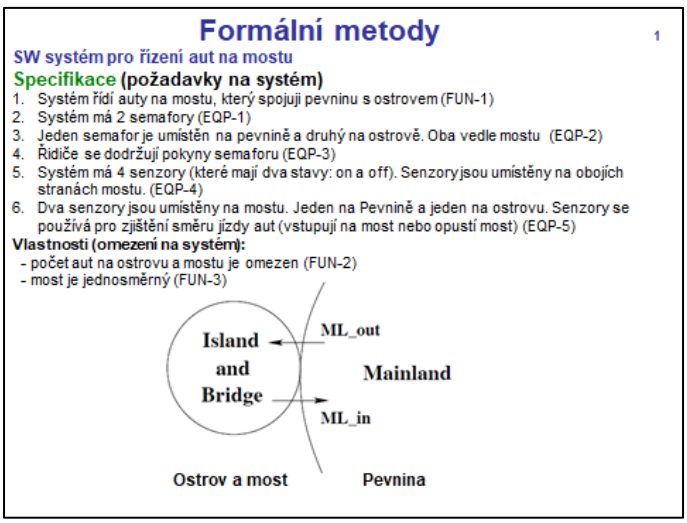
Kapitola 13. Příklad použití metody Event-B. Vývoj dependabilních systémů.

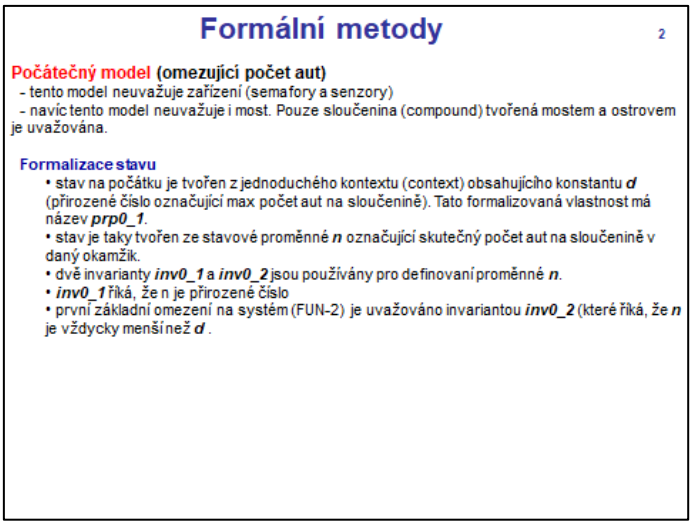
Cíl kapitoly:

- ukázat použití metody Event-B pro návrh řídícího programu
- probrat kroky upřesnění formální specifikace programu
- vysvětlit a popsat komponenty matematického modelu řídícího programu

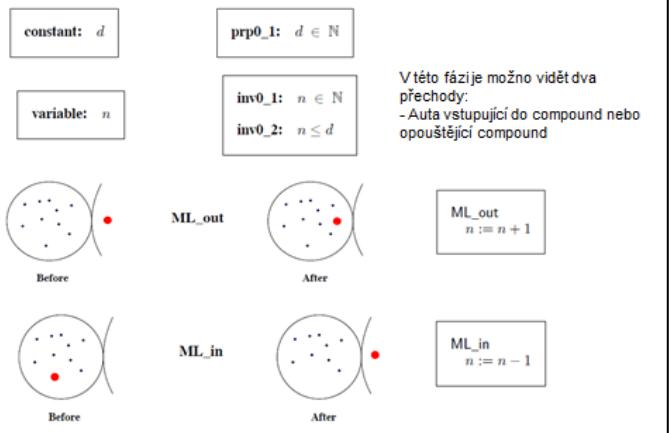
Klíčová slova: formální metody, formalizace stavu, automaty a kontexty, události, průkazové povinnosti

Výkladová část:
Následující prezentace pomůže studentům splnit úkoly samostatné práce.



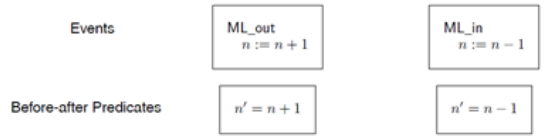


Formální metody



V této fázi je možno vidět dva přechody:
- Auta vstupující do compound nebo opouštějící compound

Formální metody



Dokazování udržení invariantů (Proving Invariant Preservation)

P(c) znamená seznam: ***P1(c), P2(c),*** . . . vlastností konstant.

Nechť v označuje proměnné a ***I(c,*** v) označuje invarianty těchto proměnn.

Nechť v' = ***E(c,*** v) je beforeafter predicate (před-a-pote predikát) spojený s touto události.

$P(c), I(c, v) \vdash I_i(c, E(c, v))$	INV

Properties Invariants ⊢ Modified Invariant	INV

The symbol ⊢ is named the *turnstile*.**(turniket)**

Formální metody

$d \in \mathbb{N}$ $n \in \mathbb{N}$ $n \leq d$ ⊢ $n + 1 \in \mathbb{N}$	ML_out / **inv0_1** / INV

$d \in \mathbb{N}$ $n \in \mathbb{N}$ $n \leq d$ ⊢ $n - 1 \in \mathbb{N}$	ML_in / **inv0_1** / INV

$d \in \mathbb{N}$ $n \in \mathbb{N}$ $n \leq d$ ⊢ $n + 1 \leq d$	ML_out / **inv0_2** / INV

$d \in \mathbb{N}$ $n \in \mathbb{N}$ $n \leq d$ ⊢ $n - 1 \leq d$	ML_in / **inv0_2** / INV

Zlepšení událostí (Improving the two Events): Uvedení Guards

ML_out **when** $n < d$ **then** $n := n + 1$ **end**

ML_in **when** $0 < n$ **then** $n := n - 1$ **end**

Formální metody

Guards označují nutné podmínky pro povolení (enable) událostí.
- Aby event ML_out byla povolena je zapotřebí aby ***n*** bylo striktně menší než ***d***.
- aby event ML_in byla povolena potřebujeme aby ***n*** bylo striktně kladné.

Tvrzení, které musí být prokázáno pomocí aplikovaní pravidla INV teď jsou modifikovány i mohou být snadno prokázány.

Formální metody

Zlepšení pravidla udržování invarianty (Improving the Invariant Preservation Rule)

$P(c)$ $I(c, v)$ $G(c, v)$ ⊢ $I_i(c, E(c, v))$	INV

Properties Invariants Guards of the event ⊢ Modified Invariant	INV

$d \in \mathbb{N}$ $n \in \mathbb{N}$ $n \leq d$ $n < d$ ⊢ $n + 1 \leq d$	ML_out / **inv0_2** / INV

$d \in \mathbb{N}$ $n \in \mathbb{N}$ $n \leq d$ $0 < n$ ⊢ $n - 1 \in \mathbb{N}$	ML_in / **inv0_1** / INV

Inicializace (Initialization)

init $n := 0$

before-after predikát je vždy after predikát. V daném případě to se týká pouze n'):

$n' = 0$

Formální metody

Prokázání absence zablokování (Deadlock Freedom Rule)

Properties Invariants ⊢ Disjunction of the guards	DLF

$d \in \mathbb{N}$ $n \in \mathbb{N}$ $n \leq d$ ⊢ $n < d \ \vee \ 0 < n$

DLF

prp0_2: $0 < d$

počet aut na ostrovu a mostu je omezen a je kladné číslo (FUN-2)

Formální metody

Závěr a shrnutí počátečného modelu

constant: d

prp0_1: $d \in \mathbb{N}$
prp0_2: $d > 0$

variable: n

inv0_1: $n \in \mathbb{N}$
inv0_2: $n \leq d$

Události (Events)

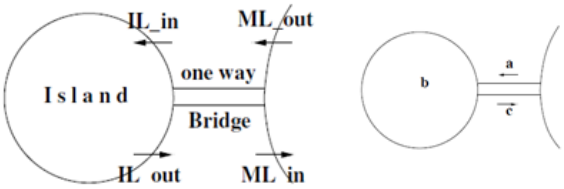
init
$n := 0$

ML_out
when
$n < d$
then
$n := n + 1$
end

ML_in
when
$0 < n$
then
$n := n - 1$
end

Formální metody

První upřesnění (First Refinement): Uvedení jednosměrného mostu



variables: a, b, c

inv1_1: $a \in \mathbb{N}$
inv1_2: $b \in \mathbb{N}$
inv1_3: $c \in \mathbb{N}$

inv1_4: $a + b + c = n$
inv1_5: $a = 0 \ \vee \ c = 0$

Formální metody

Upřesnění abstraktních událostí (Refining the Abstract Events)

(abstract_)ML_out
when
$n < d$
then
$n := n + 1$
end

(concrete_)ML_out
when
$a + b < d$
$c = 0$
then
$a := a + 1$
end

(abstract_)ML_in
when
$0 < n$
then
$n := n - 1$
end

(concrete_)ML_in
when
$0 < c$
then
$c := c - 1$
end

Properties Abstract invariant Concrete invariant Concrete guard ⊢ Modified concrete invariant	INV_REF

Formální metody

prp0_1	$d \in \mathbb{N}$
prp0_2	$0 < d$
inv0_1	$n \in \mathbb{N}$
inv0_2	$n \leq d$
inv1_1	$a \in \mathbb{N}$
inv1_2	$b \in \mathbb{N}$
inv1_3	$c \in \mathbb{N}$
inv1_4	$a + b + c = n$
inv1_5	$a = 0 \ \vee \ c = 0$
Concrete guards of ML_out	$a + b < d$
	$c = 0$
⊢	⊢
Modified Invariant **inv1_4**	$a + 1 + b + c = n + 1$

ML_out / **inv1_4** / INV_REF

Formální metody

prp0_1	$d \in \mathbb{N}$
prp0_2	$0 < d$
inv0_1	$n \in \mathbb{N}$
inv0_2	$n \leq d$
inv1_1	$a \in \mathbb{N}$
inv1_2	$b \in \mathbb{N}$
inv1_3	$c \in \mathbb{N}$
inv1_4	$a + b + c = n$
inv1_5	$a = 0 \ \vee \ c = 0$
Concrete guards of ML_in	$0 < c$
⊢	⊢
Modified Invariant **inv1_5**	$a = 0 \ \vee \ c - 1 = 0$

ML_in / **inv1_5** / INV_REF

Upřesnění události Init

init
$a, b, c := 0, 0, 0$

$a' = 0 \ \wedge \ b' = 0 \ \wedge \ c' = 0$

Formální metody

Uvedení nových událostí (Introducing New Events)

IL_in
when
$0 < a$
then
$a := a - 1$
$b := b + 1$
end

IL_out
when
$0 < b$
$a = 0$
then
$b := b - 1$
$c := c + 1$
end

Prokázání konvergence nových událostí (Proving the Convergence of the New Events)

prp0_1	$d \in \mathbb{N}$
prp0_2	$0 < d$
inv0_1	$n \in \mathbb{N}$
inv0_2	$n \leq d$
inv1_1	$a \in \mathbb{N}$
inv1_2	$b \in \mathbb{N}$
inv1_3	$c \in \mathbb{N}$
inv1_4	$a + b + c = n$
inv1_5	$a = 0 \ \vee \ c = 0$
Concrete guards of IL_in	$0 < a$
⊢	⊢
Modified variant < Variant	$2 * (a - 1) + b + 1 < 2 * a + b$

variant_1 : $2 * a + b$

IL_in / WFD_REF2

Formální metody

prp0_1
prp0_2
inv0_1
inv0_2
inv1_1
inv1_2
inv1_3
inv1_4
inv1_5
Concrete guards of IL_out

⊢

Modified variant < Variant

$d \in \mathbb{N}$
$0 < d$
$n \in \mathbb{N}$
$n \leq d$
$a \in \mathbb{N}$
$b \in \mathbb{N}$
$c \in \mathbb{N}$
$a + b + c = n$
$a = 0 \ \vee \ c = 0$
$0 < b$
$a = 0$

⊢

$2 * a + b - 1 < 2 * a + b$

IL_out / WFD_REF2

Formální metody

Absence zablokování

Properties
Abstract invariants
Concrete invariants
Disjunction of abstract guards

⊢

Disjunction of concrete guards

DLF_REF

prp0_1
prp0_2
inv0_1
inv0_2
inv1_1
inv1_2
inv1_3
inv1_4
inv1_5

⊢

Disjunction of concrete guards

$d \in \mathbb{N}$
$0 < d$
$n \in \mathbb{N}$
$n \leq d$
$a \in \mathbb{N}$
$b \in \mathbb{N}$
$c \in \mathbb{N}$
$a + b + c = n$
$a = 0 \ \vee \ c = 0$

⊢

$(a + b < d \ \wedge \ c = 0) \ \vee$
$c > 0 \ \vee$
$a > 0 \ \vee$
$(b > 0 \ \wedge \ a = 0)$

DLF_REF

Formální metody

Shrnutí prvního upřesnění

constants: d
variables: a, b, c

inv1_1: $a \in \mathbb{N}$
inv1_2: $b \in \mathbb{N}$
inv1_3: $c \in \mathbb{N}$

inv1_4: $a + b + c = n$
inv1_5: $a = 0 \ \vee \ c = 0$
variant1: $2 * a + b$

ML_in
when
$0 < c$
then
$c := c - 1$
end

ML_out
when
$a + b < d$
$c = 0$
then
$a := a + 1$
end

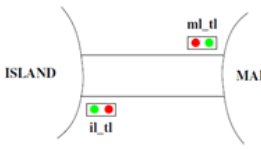
IL_in
when
$0 < a$
then
$a := a - 1$
$b := b + 1$
end

IL_out
when
$0 < b$
$a = 0$
then
$b := b - 1$
$c := c + 1$
end

init
$a, b, c := 0, 0, 0$

Formální metody

Druhé upřesnění: Uvedení semaforů



Upřesnění stavu

carrier sets: $COLOR$
constants: $d, red, green$

prp2_1: $COLOR = \{green, red\}$
prp2_2: $green \neq red$

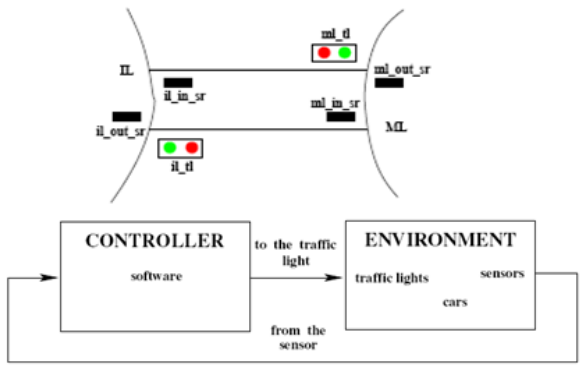
variables: a, b, c, ml_tl, il_tl

inv2_1: $ml\_tl \in COLOR$
inv2_2: $il\_tl \in COLOR$
inv2_3: $ml\_tl = green \ \Rightarrow \ a + b < d \ \wedge \ c = 0$
inv2_4: $il\_tl = green \ \Rightarrow \ 0 < b \ \wedge \ a = 0$

Note again that invariants inv2_3 and inv2_4 are *conditional invariants. Such invariants are introduced* by means of the logical implication operator "$\Rightarrow$".

Formální metody

Třetí upřesnění : Uvedení senzorů



Formální metody

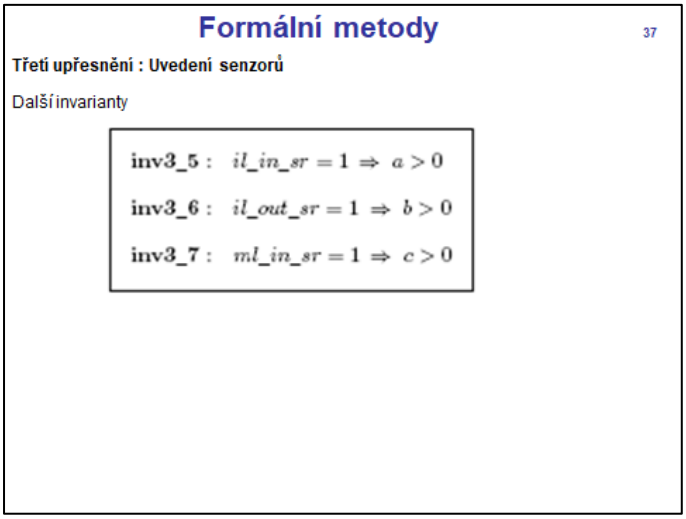
Třetí upřesnění : Uvedení senzorů

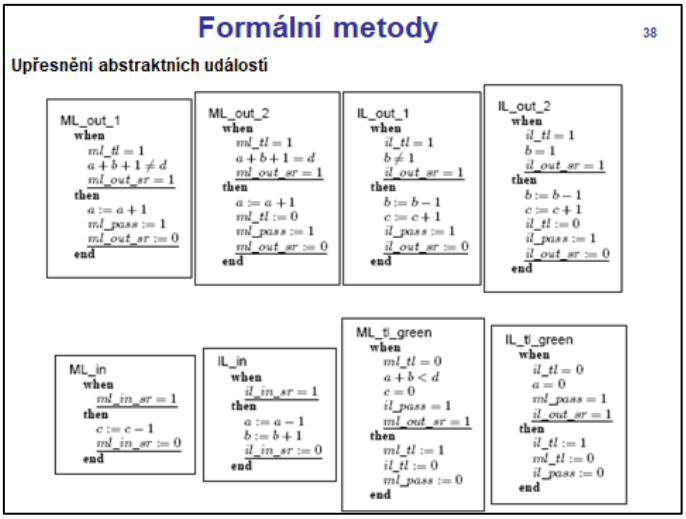
Je nutno rozšířit stav modelu pomocí uvedení 4 nových proměnných, které korespondují každému stavu senzoru:
ml_out_sr, ml_in_sr, il_out_sr, a il_in_sr.
Tyto proměnné jsou definovány invariantami **inv3_1 až inv3_4**:

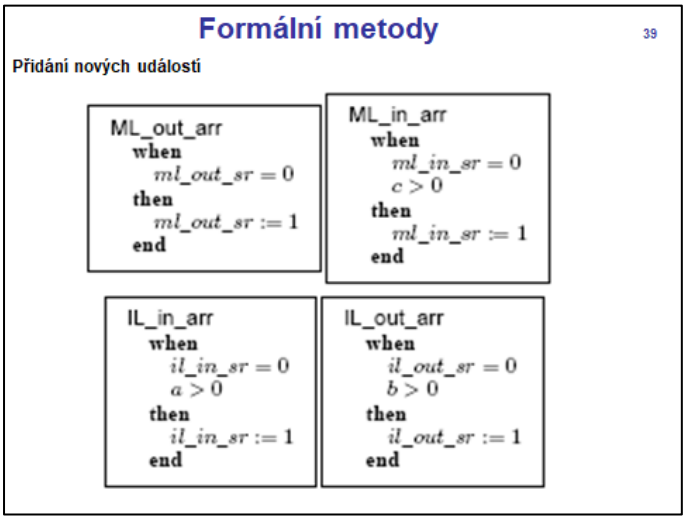
constants : d

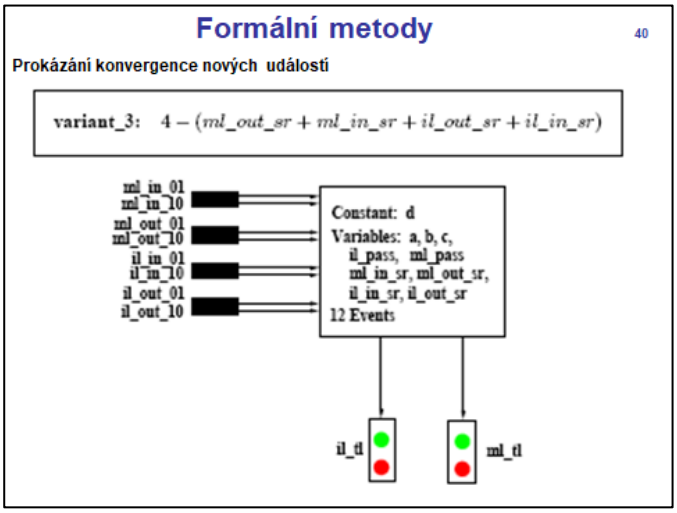
variables : $a, b, c,$
ml_tl, il_tl
ml_pass, il_pass
$ml\_out\_sr, ml\_in\_sr,$
il_out_sr, il_in_sr

inv3_1 : $ml\_out\_sr \in \{0, 1\}$
inv3_2 : $ml\_in\_sr \in \{0, 1\}$
inv3_3 : $il\_out\_sr \in \{0, 1\}$
inv3_4 : $il\_in\_sr \in \{0, 1\}$









Kontrolní otázky:

- co zahrnuje událost?
- kreré důkazové povinnosti byly použity v uvažovaném příkladu?
- co označuje Guard (straž) pro událost?

Úkoly pro samostatnou práci:
1) Prostudovat uvedený příklad. Použit obsah výkladové části.

Kapitola 14. Dependabilita s ohledem na závady způsobené svévolnými činnostmi

Cíl kapitoly:

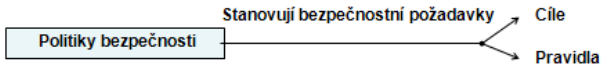
- charakterizovat politiky bezpečnosti
- probrat bezpečnostní vlastnosti
- probrat bezpečnostní selhání
- vysvětlit metody zajišťující bezpečnost systému

Klíčová slova: politiky bezpečnosti, bezpečnostní vlastnosti, bezpečnostní selhání, intrusion (tj. nežádoucí vniknutí), útok, zranitelnost

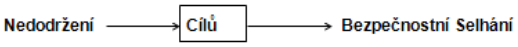
Výkladová část:
Následující prezentace pomůže studentům splnit úkoly samostatné práce.

Dependability s ohledem na závady způsobené svévolnými činnostmi

Politiky bezpečnosti



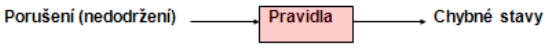
Cíle se snaží zachytit bezpečnostní požadavky vysoké úrovně.
Typické bezpečnostní cíle mohou zahrnovat:
- Důvěrnost citlivých Dat musí být udržována;
- Integrita a Dostupnost systémových Dat pro oprávněné uživatele musí být udržována.

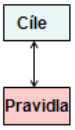


Pravidla jsou omezení (na systém) nižší úrovně (t.j. stavů a přechodů).
■ Pravidla jsou skonstruovaná (navržená) tak aby zajistit, že systém je robustný (t.j. Odolný proti svévolným činnostem).
Některá pravidla budou aplikovaná na uživatele (pro omezení činnosti uživatelů) i vystupují jako:
♦ *prohibice* ♦ *povinnosti* ♦ *závazky*
Některá pravidla budou aplikovaná na technický systém i vystupují jako:
♦ *pravidla řízení přístupy nebo výsadami (privilegiumem)*

Dependability s ohledem na závady způsobené svévolnými činnostmi

Politiky bezpečnosti





Ideálně: dodržení pravidel garantuje, že cíle budou také dodrženy.

V praxi: některé bezpečnostní cíle je nemožné redukovat na pravidla.

Bezpečnostní vlastnosti

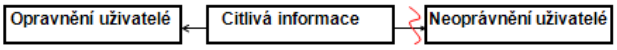
1. Důvěrnost
2. Integrita
3. Dostupnost

Dependability s ohledem na závady způsobené svévolnými činnostmi

Důvěrnost

Důvěrnost (diskrétnost) je vlastnost, že neoprávnění uživatelé nezískají utajované informace.

Důvěrnost je obvykle je popsána v termínech řízení tokem informací.



Politika bude určovat, která proudění jsou dovolena a která jsou zakázaná.

Ztráta Důvěrnosti (ZD)
■ ZD znamená, že některé informace se nachází v místě kde nesmí být
■ Odhalení ZD může být obtižné
■ Obnovení ZD je velmi těžké

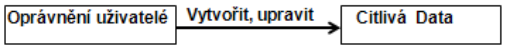
ZD je Chyba (t.j. část stavu systému, co může vést k selhání).
Selhání (v tom případě) znamená, že informace se objeví vně uvažovaného systému.

Dependability s ohledem na závady způsobené svévolnými činnostmi

Integrita

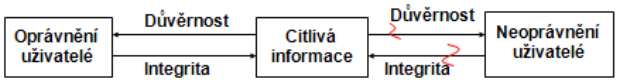
Integrita je vlastnost, že Data (údaje) se zůstávají "Validní"

V praxi, Integrita je často nahrazená pracovními omezeními: záznamy mohou být vytvořené a upravené pouze oprávněnými uživateli.



Což posouvá problém definice "Validnosti" dat do problému, který záleží na poctivosti a kompetenci oprávněných uživatelů.

Tudíž, Důvěrnost vyžaduje aby utajovaná informace neproudila k neoprávněným uživatelům.
Zatímco, Integrita vyžaduje aby neoprávnění uživatelé neovlivňovaly utajované informace.



Dependability s ohledem na závady způsobené svévolnými činnostmi

Dostupnost
Důvěrnost a Integrita jsou o zabránění výskytů nežádoucích událostí.
Dostupnost je o zajištění toho aby se žádoucí události vyskytovaly.

Jsou tam dva aspekty, které se týkají Dostupnosti:
- hodnotná doména;
- časová doména.

Ostatní bezpečnostní vlastnosti
• Anonymita;
• Zachování soukromí;
• Autentičnost (pravost);
• Zodpovědnost;
• Nepopíratelnost, atd.
mohou být definované v termínech Důvěrnosti, Integrity a Dostupnosti.

Dependability s ohledem na závady způsobené svévolnými činnostmi

Meta-informace:
❑ Termín poskytování služby, nebo stvoření, modifikace, nebo destrukce (zničení) informační jednotky.
❑ Identita osoby, která vyvolává operace.
❑ Umístění nebo adresa informační jednotky, komunikační entity, zařízení.
❑ Úroveň utajovanosti informační jednotky nebo meta-informace.
❑ Jistota nebo úroveň pravděpodobnosti informační jednotky nebo meta-informace.

Sledovácí vlastnosti

Sledovácí/bezpečnostní vlastnost definuje tato chování, která jsou považovaná za přijatelná, nebo bezpečná, a tím pádem mohou být vyjádřeny jako množina stop.

Chovaní systému jsou považovaná za Stopy.

- Stopa systému může být chápaná jako záznam určitého pozorování konkrétního vykonání systému.

Záznam › posloupnosti akcí a stavů.

Aby ověřil, že systém vyhovuje daným bezpečnostním vlastnostem postačí vyšetřit jestli-že množina stop $\tau(S)$ systému je podmnožina množiny bezpečnostních stop.

Aby zkontroloval, že systém udržuje bezpečnostní vlastnosti v průběhu fungování, je nutné monitorovat jeho akce a ověřit, že stopy zůstávají uvnitř množiny bezpečnostních stop.

Monitor provádění: je konceptuální zařízení, které běží paralelně se systémem.
MP pozoruje chování systému a blokuje akce systému, které by mohly vést k porušení sledovací vlastnosti. [1]

Pojem Selhání (používaný v Dependability) přirozeně zapadá do vlastností systému, založených na sledování.

Selhání s ohledem na nesledovací vlastnosti může být charakterizované pouze konceptuálně i nemůže být přímým výsledkem pozorování systému.

Selhání v zajištění bezpečnosti (bezpečnostní selhání)

Příčiny selhání t.j. Závady:

1) Závady ve specifikaci cílů (bezpečnosti);
2) Závady ve specifikaci pravidel (bezpečnosti);
3) Závady v implementace technických pravidel;
4) Závady v technických mechanismech nízké úrovně;
5) Závady v sociálně-technických mechanismech; atd.

1. Závady ve specifikaci cílů.
   Jsou to závady kvůli nesprávnému popisu (zahrnutí) bezpečnostních požadavků.

2. Závady ve specifikaci pravidel.
   Může být, že pravidla nebudou úplně odpovídat cílům. To je možné s různých důvodů:
   - Pravidla mohou být nekompletní;
   - Pravidla mohou být nekonsistentní nebo nejednoznačná;
   - Chyby mohou být v analýze logických následcích dané množiny pravidel.

Bezpečnostní selhání

3. Závady v implementace technických pravidel.
   To je možné z následujících důvodů:
   - jednoduché chyby v kódování (implementovaná pravidla neodpovídají abstraktně specifikovaným pravidlům);
   - negarantovaná integrita pravidel;
   - chybná údržba pravidel;
   - chyby v mapování pravidel na architekturu, atd.

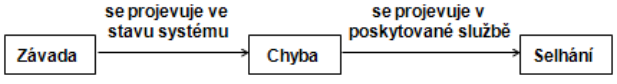
4. Závady v nízko-úrovňových technických mechanismech.
   Mechanismy nízké úrovně:
   - autentifikace (ověření pravosti),
   - zašifrování , atd.

5. Závady v sociálně-technickém mechanismu.
   - povinnosti a závazky mohou být nepřesně nebo nedostatečně formulované;
   - uživatelé mohou nedodržet svoje povinnosti a závazky.

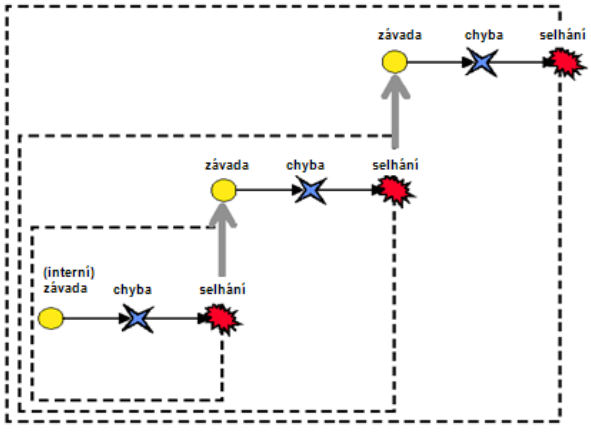
Model Závady

Příčinný řetězec implementací

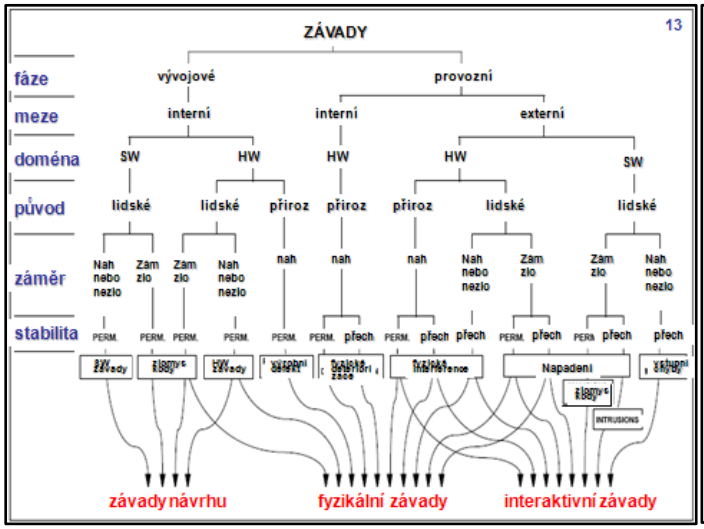
Závada: je posouzená nebo hypotetická příčina chyby;
Chyba: je takový stav systému, který může vést k selhání;
Selhání: poskytovaná služba se liší od správné služby.



Kvůli rekurzivní definici systému, selhání na dané úrovni dekompozice může být přirozeně interpretované jako závada na další vyšší úrovni dekompozice, což vede k hierarchickému příčinnému řetězci.



Hierarchický příčinný řetězec implementací

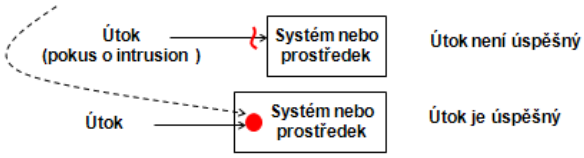


Dependability s ohledem na závady způsobené svévolnými činnostmi

Intrusion (nežádoucí vniknutí), Útok a Zranitelnost

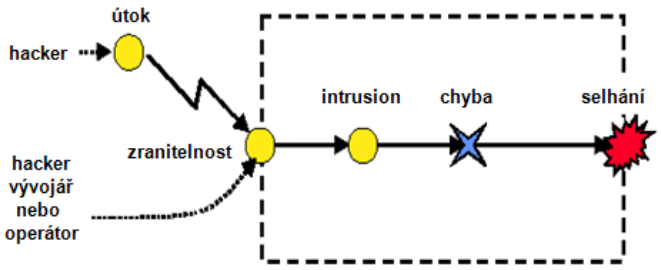
Intrusion: záměrné-zlomyslné SW operační závady, které pochází zvenku systému.
Alternativa ⟹ Útok (napadení)

Útok je pokus o intrusion.
Intrusion je výsledek útoku, který byl úspěšný.



Dependability s ohledem na závady způsobené svévolnými činnostmi

Jsou tam dvě příčiny pro vznik Intrusion:
1) Zlomyslný čin nebo Útok, který se pokusí o využití slabiny v systému.
2) Nejméně jedna slabina, nedostatek (vada) nebo zranitelnost.



Dependability s ohledem na závady způsobené svévolnými činnostmi

- Zranitelnosti mohou vzniknout během vývoje systému (vývojář) nebo během provozu systému (operátor).

- *Zranitelnosti* mohou vzniknout
  - náhodně (vývojář, operátor) nebo záměrně (hacker),
  - se zlým úmyslem (hacker) nebo bez zlého úmyslu (vývojář, operátor).

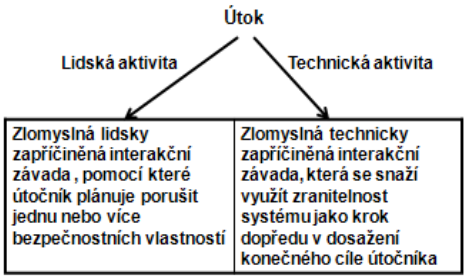
Technologie (metody) odolnosti proti intrusion mají za cíl tolerovat tuto událost, že zranitelnosti mohou být úspěšně využité útočníkem.

Příklady intrusions jsou interpretované v termínech zranitelnosti a útoku:
1. Outsider (cizí osoba) proniká do systému pomocí odhalení hesla uživatele; Zranitelnost spočívá v konfiguraci systému, ve špatně vybraném hesle (např. příliš krátké, nebo snadno podléhající slovníkovému útoku).
2. Insider (interní pracovník) zneužívá svoje privilegium (zneužívá svoje práva); Zranitelnost spočívá v specifikaci nebo v sociálně-technickém návrhu systému (např. nedodržení principu „nejmenší privilegium“, nedostatečné prověření zaměstnanců).
3. Outsider využívá „sociální inženýrství“ (napr. podplácení) aby přinutil insidera zneužít svoje privilegium ve prospěch outsidera; Zranitelnost je přítomnost podpláceného insidera , což je právě výsledkem špatného sociálně-technického návrhu systému (např. Nedostatečné prověření zaměstnanců).

Typy závad (pravděpodobné příčiny chyb)

Útok	Zranitelnost	Intrusion
Zlomyslná interakční závada pomocí které útočník plánuje porušit jednu nebo více bezpečnostních vlastností: pokus o Intrusion	Závada vytvořená v průběhu vývoje systému nebo během jeho provozu i může být využitá aby způsobila Intrusion	Zlomyslná závada způsobená zvenku, která je výsledkem útoku, který úspěšně využil zranitelnost systému

Dependability s ohledem na závady způsobené svévolnými činnostmi



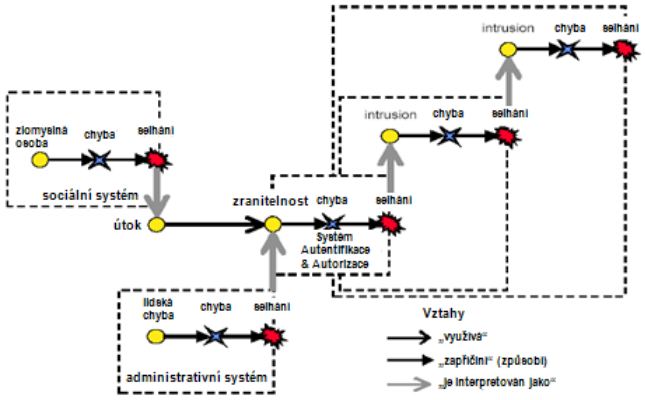
Když Technický útok je spáchán ve prospěch útočníka pomocí škodlivého kódu považujeme takový kód za Agenta útočníka.

Škodlivé kódy → Záměrně instalované zranitelností (např. zadní vrátka - trapdoor)
Škodlivé kódy → Agenty útočníka

Šíření	- Samo-šíření (např. vir nebo červ); - Bez šíření
Podmínky spouštění	- Trvale aktivované (Trojský kůň); - Aktivované šťastnou náhodou nic netušící obětí (Trojský kůň); - Ostatní podmínky (určitý čas, vstupní hodnota, atd.) (bomba nebo Zombie)
Terč útoku	- Lokální (bomba nebo Trojský kůň); - Vzdálená (Zombie)
Cíl útoku	- Odhalení (důvěrnost); - Změna (integrita); - Odmítnutí služby (dostupnost)

Šíření intrusion

■ Bezpečnostní selhání na jedné úrovni dekompozice systému může být interpretované jako intrusion na další vyšší úrovni.



Dependability s ohledem na závady způsobené svévolnými činnostmi

Oblastí vniknutí (intrusion)

Fault containment region (min Oblast obsahující závadu) [4] může být definována jako množina komponentů, které selhaly:
1) Jako Atomická jednotka
2) Statisticky nezávislým způsobem s ohledem na ostatní Oblasti OZ.

Předpoklady:
- Chování závadné OOZ je neomezené;
- Jsou tam omezený počet závadných OOZ v uvažovaném Systému.

Systém odolný proti Intrusion má za cíl garantovat určité bezpečnostní vlastnosti navzdory skutečnosti, že některé komponenty systému mohou být kompromitované buď úplatným (zkorumpovaným) administrátorem systému nebo zkorumpovanými uživateli.

Příklad: Oblast vniknutí (Intrusion) ⇨ Místo přístupu (Přístupový bod)

Garance bezpečnosti mohou být dané pouze vzhledem k množině nezkorumpovaných přístupových bodů (např. přístupové body A, B, C, D, G and H). Pro nezkorumpované uživatele nezkorumpovaných přístupových bodů systém odolný proti intrusion musí být schopen zajistit garance, které se týkají důvěrnosti, integrity a dostupnosti dat, která vlastní tito uživatelé. Garance musí být zajištěné navzdory tomu, že je tam určitý počet zkorumpovaných komponentů, administrátorů nebo uživatelů systému. [5]

Dependability s ohledem na závady způsobené svévolnými činnostmi

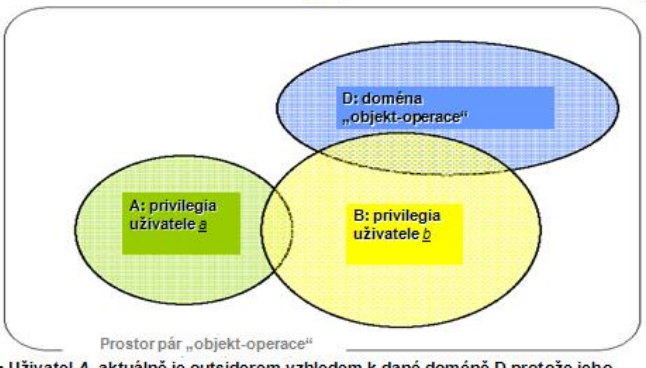
■ Doména obsahující Intrusion může být interpretována v termínech množiny práv, která se útočníkovi podařilo získat (tj. doména aktuálních privilegií).
■ Doména aktuálních privilegií útočníka definuje míru kontroly, kterou má útočník nad systémem. Útočník může zlomyslně zneužít kterýkoli pár: Objekt-operace v rámci této domény privilegií. Takže celá doména může být považovaná jako zkorumpovaná.

Privilegia (krádež a zneužívání)

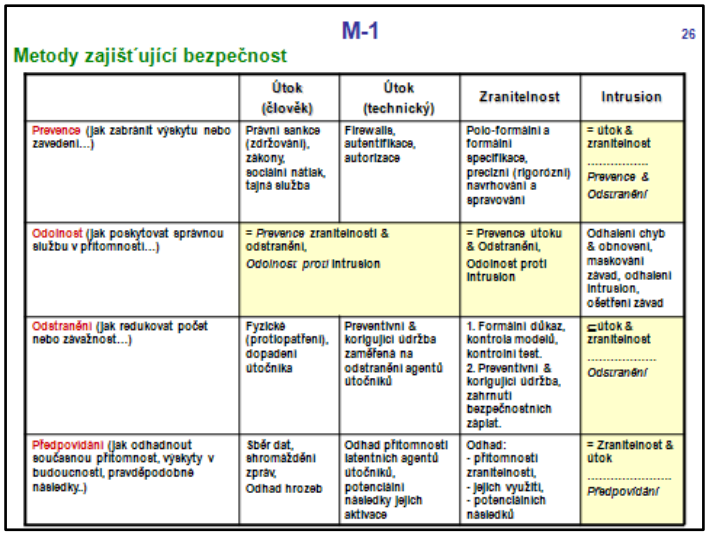
Osoba má práva na objekt uvnitř systému pouze když je oprávněná provádět specifikované operace na objektu. Tím pádem Práva mohou být vyjádřená jako pár „Objekt-operace“.

Množina Práv osoby je jeho Privilegií.

Osoba může spáchat (dopustit se):
- Krádež Privilegií: neoprávněné zvýšení privilegií (tj. změna v privilegii uživatele, která není povolená Politikou řízení přístupů k systému.
- Zneužívání privilegií: zneužití práva (tj. neoprávněné použití oprávněných operací).



■ Uživatel *A* aktuálně je outsiderem vzhledem k dané doméně D protože jeho privilegia se nekříží s touto doménou. Uživatel *A* může působit uvnitř domény D pouze když odcizí privilegium, který je nad jeho aktuálními privilegii.
■ Uživatel *B* je insider (interní pracovník) vzhledem k doméně D, ale je outsiderem vzhledem k poddoméně D-B. Uživatel *B* může spáchat obojí druhy intrusion v doméně D: zneužít privilegium uvnitř D∩B nebo odcizit privilegium s D-B.



M-1 26

Metody zajišťující bezpečnost

	Útok (člověk)	Útok (technický)	Zranitelnost	Intrusion
Prevence (jak zabránit výskytu nebo zavedení...)	Právní sankce (zdržování), zákony, sociální nátlak, tajná služba	Firewalls, autentifikace, autorizace	Polo-formální a formální specifikace, precizní (rigorózní) navrhování a spravování	= útok & zranitelnost --- *Prevence & Odstranění*
Odolnost (jak poskytovat správnou službu v přítomnosti...)	= Prevence zranitelnosti & odstranění, *Odolnost proti* intrusion		= Prevence útoku & Odstranění, Odolnost proti intrusion	Odhalení chyb & obnovení, maskování závad, odhalení intrusion, ošetření závad
Odstranění (jak redukovat počet nebo závažnost...)	Fyzické (protlopatření), dopadení útočníka	Preventivní & korigující údržba zaměřena na odstranění agentů útočníků	1. Formální důkaz, kontrola modelů, kontrolní test. 2. Preventivní & korigující údržba, zahrnutí bezpečnostních zápist.	⊆útok & zranitelnost --- *Odstranění*
Předpovídání (jak odhadnout současnou přítomnost, výskyty v budoucnosti, pravděpodobné následky.)	Sběr dat, shromáždění zpráv, Odhad hrozeb	Odhad přítomnosti latentních agentů útočníků, potenciální následky jejich aktivace	Odhad: - přítomnosti zranitelnosti, - jejich využití, - potenciálních následků	= Zranitelnost & útok --- *Předpovídání*

Kontrolní otázky:

- charakterizujte politiky bezpečnosti
- definujte bezpečnostní vlastnosti
- charakterizujte bezpečnostní selhání
- definujte intrusion, útok a zranitelnost

Úkoly pro samostatnou práci:
1) Prostudovat politiky bezpečnosti, bezpečnostní vlastnosti, bezpečnostní selhání a metody zajišťující bezpečnost. Použit prezentaci ve výkladové části a literaturu [1].
2) Vytvořit tabulku s příklady bezpečnostních selhání různých systémů. Pro každý příklad popsat závadu (intrusion, útok, zranitelnost), chybu a selhání systému. Tabulka má mít 5 řádků (jeden řádek pro každý příklad) a 4 sloupce. První sloupec pro popis systému, druhý pro popis závady, třetí sloupec pro popis chyby a čtvrtý sloupec pro popis selhání systému.

Literatura

1. Mashkov V., Fišer J. Samokontrola a samodiagnostika na systémové úrovni. *Ukrainian Academic Press*, Lviv, 2010, 176 stran.
2. Laprie, J.C., ed. Dependability: Basic concepts and terminology- in English, French, German, Italian and Japanese. *Springer-Verlag*, Vienna, Austria, 1992.
3. Laprie, J.C. Dependability of software-based critical systems*: in Dependable Network Computing*. *Kluwer Academic Publishers*, 1999.
4. Preparata F., Metze G., Chien R. On the connection assignment problem of diagnosable system. *IEEE Trans. on Electronic Computers.* Vol. EC-16, No. 12, 1967, pp. 848-854.
5. Barsi T., Grandoni T., Maestrini P. A theory of diagnosability of digital systems. *IEEE Trans. on Comp.* Vol. C-25, No. 6, 1976, pp. 585-593.
6. Vedeshenkov V. On organization of self-diagnosable digital systems. *Automation and computer engineering*. Vol. 7, 1983, pp. 133-137.
7. Fujiwara H., Kinoshita K. Some existence theorems for probabilistically diagnosable systems. *IEEE Trans. on Comp*. Vol. C-27, No. 4, 1981, pp. 297-303.
8. Mallela S., Masson G. Diagnosable systems for intermittent faults. IEEE Trans. on Comp. Vol. C-27, 1978, pp. 379-384.
9. Mashkov V. Selected problems of system level self-diagnosis. Lviv: Ukrainian Academic Press, 2011. ISBN 978-966-322-365-0.
10. Avizienis, J.-C. Laprie and Brian Randell: Fundamental Concepts of Dependability. Research Report No 1145, Lydford g DrAAS-CNRS, April 2001
11. J.R. Abrial. Modeling in Event-B: System and Software Engineering. Cambridge University Press, 2010, ISBN 978-0-521-89556-9.