4. NÁSTROJE PRO TVORBU ESENCIÁLNÍHO MODELU V YOURDONOVĚ STRUKTUROVANÉ METODĚ

Současné strukturované techniky návrhu systému zahrnují nástroje a postupy pro **tvorbu esenciálního modelu informačního systému**:

- funkční model (vyjádřený pomocí DFD diagramů a minispecifikací elementárních funkcí);
- datový model (vyjádřený pomocí ERD diagramu a konceptuálního schématu dat);
- model přechodu stavů systému (vyjádřený pomocí STD diagramů);

a pro **implementační model systému**:

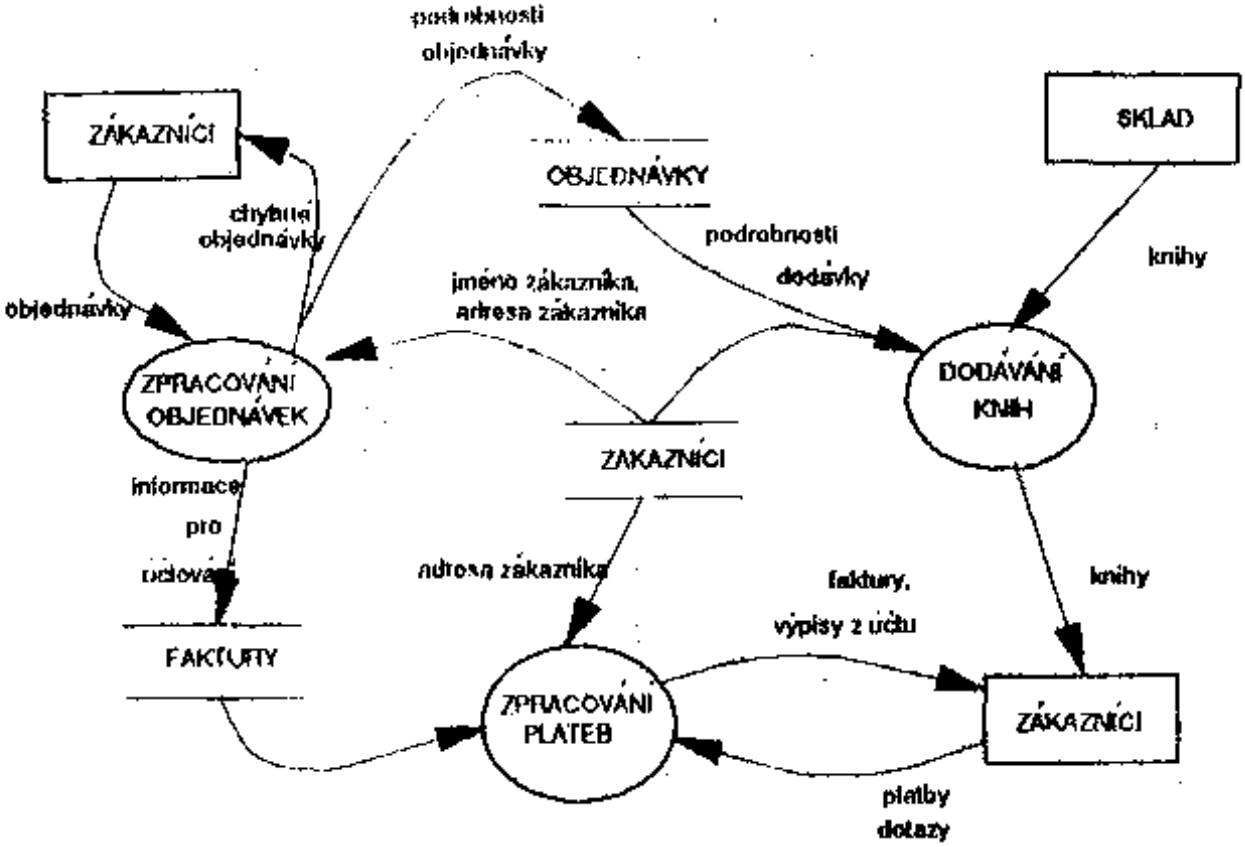
- model architektury programových systémů (vyjádřené pomocí Structure Chart a specifikace modulů).

Všechny tyto modely musí být vzájemně provázané a konzistentní. Nedílnou součástí návrhu je slovník dat (Data Dictionary), v němž jsou obsaženy všechny doplňující potřebné údaje o datech systému.

4.1. Diagram datových toků (DFD)

4.1.1. Co jsou DFD? - Definice DFD

DFD je zkratka z anglického označení "data flow diagram" (diagram datových toků). Slouží jako grafický prostředek návrhu a zobrazení funkčního modelu systému. Je ve formě sítě, pomocí níž se vyjadřují funkce (komponenty) informačního systému a některé jejich vztahy. Úvodní příklad je na obr. 1.



Obr. 1. DFD - Úvodní příklad použití diagramu

Pro DFD se používají i názvy:

- bublinový diagram (pozor, neplést s bubble chart pro datovou analýzu),
- procesní model,
- funkční model,
- work/flow diagram.

4.1.2. Historie vzniku DFD

První "DFD" diagram vznikl ve 20. letech našeho století. V jedné organizaci, kde pracovalo mnoho úředníků, byl najat konzultant, aby organizaci reorganizoval a navrhl zjednodušení její práce. Tento konzultant každého úředníka nakreslil jako bublinu a mezi úředníky nakreslil pro každý doklad, který si předávali, jednu šipku. Úředníky, kteří měli mezi sebou hodně šipek, posadil do jedné kanceláře. Ti, kteří spolu komunikovali málo seděli zvlášť. Tak vznikl první diagram toku dokumentů, první předchůdce DFD.

Pro vývoj informačních systémů (IS) a automatizovaných IS se DFD začaly používat v 70. letech. Jejich základ je v SADT (Structured Analysis Design Technique) v "activity diagrams" (r. 1975). Od té doby se používají v řadě technik, vyvíjí se jejich syntax i sémantika:

- 1977 Gane a Searson: "Structured systems analysis
- Tools and Techniques" pro návrh systému shora dolů, podle datových toků,
- 1978 De Marco: "Structured Analysis and System Specification". Klasický přístup k návrhu systému. Nejprve se vytváří současný fyzický model. Z něho se abstrakcí odvodí současný logický model. Vývoj nového informačního systému má opět dvě etapy: tvorbu nového logického modelu a potom tvorbu nového fyzického modelu. Pro tento klasický přístup k návrhu systému je typická dlouhá doba vývoje a absence datového modelování (zahrnuty pouze funkční specifikace),
- 1979 Yourdon a Constantin: "Structured Design",
- 1984 Yourdon: "Essential Systems Analysis",
- 1985 Ward/Mellor: "Structured Techniques for Real Time Systems" - přidávají do DFD prvky řízení,
- 1989 Yourdon: "Modern Structured Analysis" - propojuje funkční a datové modelování. Vzniká koncept modelování systému dle událostí. Rozlišují se úrovně modelů: model prostředí, model chování a implementační model. V poslední době se objevuje snaha využít prvků objektového přístupu.

DFD dle De Marca vyjadřuje toky dat a jejich transformace - která data a jaké procesy jsou potřeba k vyprodukování příslušných dat. DFD nevyjadřují časové uspořádání procesů. Návaznost mezi procesy je datová (s výjimkou řídících toků a transformací, které budou vysvětleny později). Procesy, jejichž výstupní datové toky jsou vstupem do jiného procesu, musí tyto toky vyprodukovat dříve, než je přijímající procesy zpracují. Nelze prostě přečíst pf-klíč, který ještě nebyl zmáčknut.

Nejdůležitějšími zásadami, které umožňují udržet konzistenci modelu funkcí, jsou: zjemňování procesů shora dolů, jednoznačná jména procesů a číslování dle úrovní.

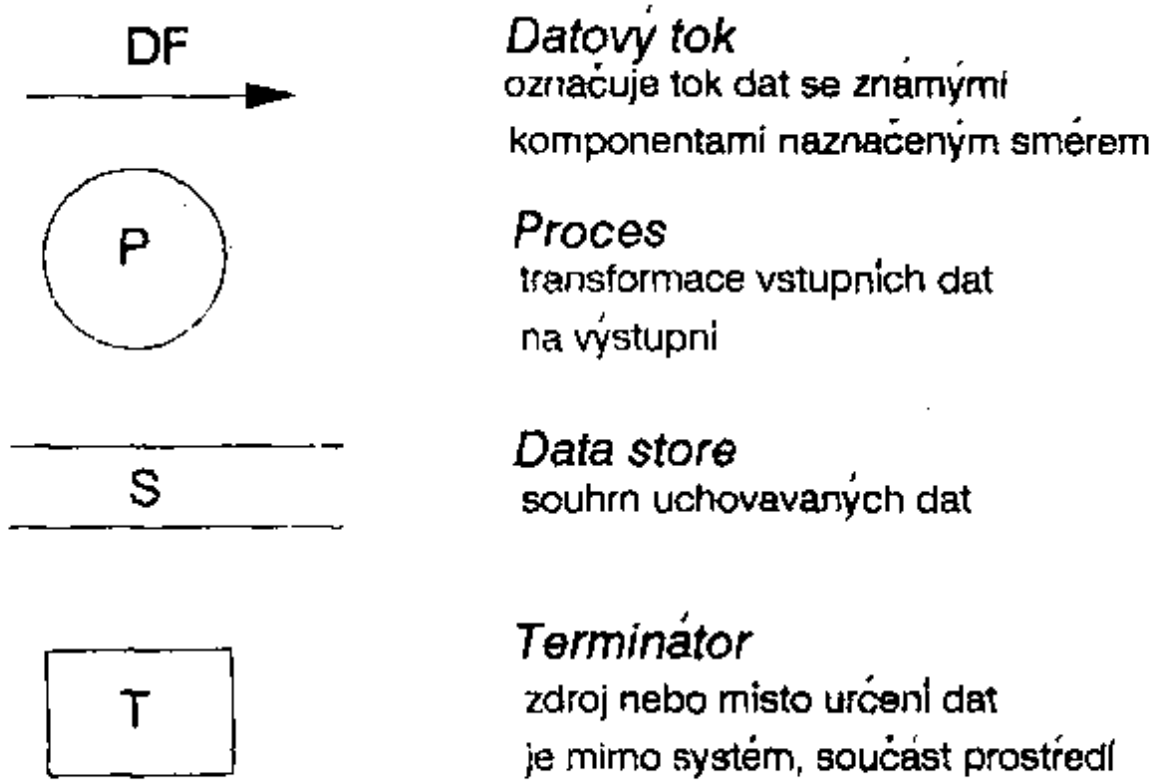
Jaké jsou problémy De Marcových diagramů a historicky výchozích fází Yourdona? Systém se modeluje pouze pomocí DFD, a tudíž je základ systému pouze ve funkcích. Chybí informace o vztazích mezi daty a o vlivu událostí, probíhajících v čase (viz problémy klasických strukturovaných přístupů).

S rozvojem aplikací zpracování v reálném čase došlo k dynamizaci DFD (doplnění o řídící transformace a toky). S rozvojem datového modelování, databázových systémů apod. došlo k propojování DFD s datovým mode-

lováním a datovou analýzou (funkce versus data ≈ různé úhly pohledu na jeden systém).

4.1.3. Prvky (symboly) a syntax DFD

V DFD se používají jako základní symboly uvedené na obr. 2.



Obr. 2. DFD - Základní symboly používané v DFD

PROCES (transformace, funkce)

provádí transformací dat, která vede k vyprodukování výstupu (transformace vstupu na výstup). Značí se kolečkem (zaobleným obdélníkem, elipsou).

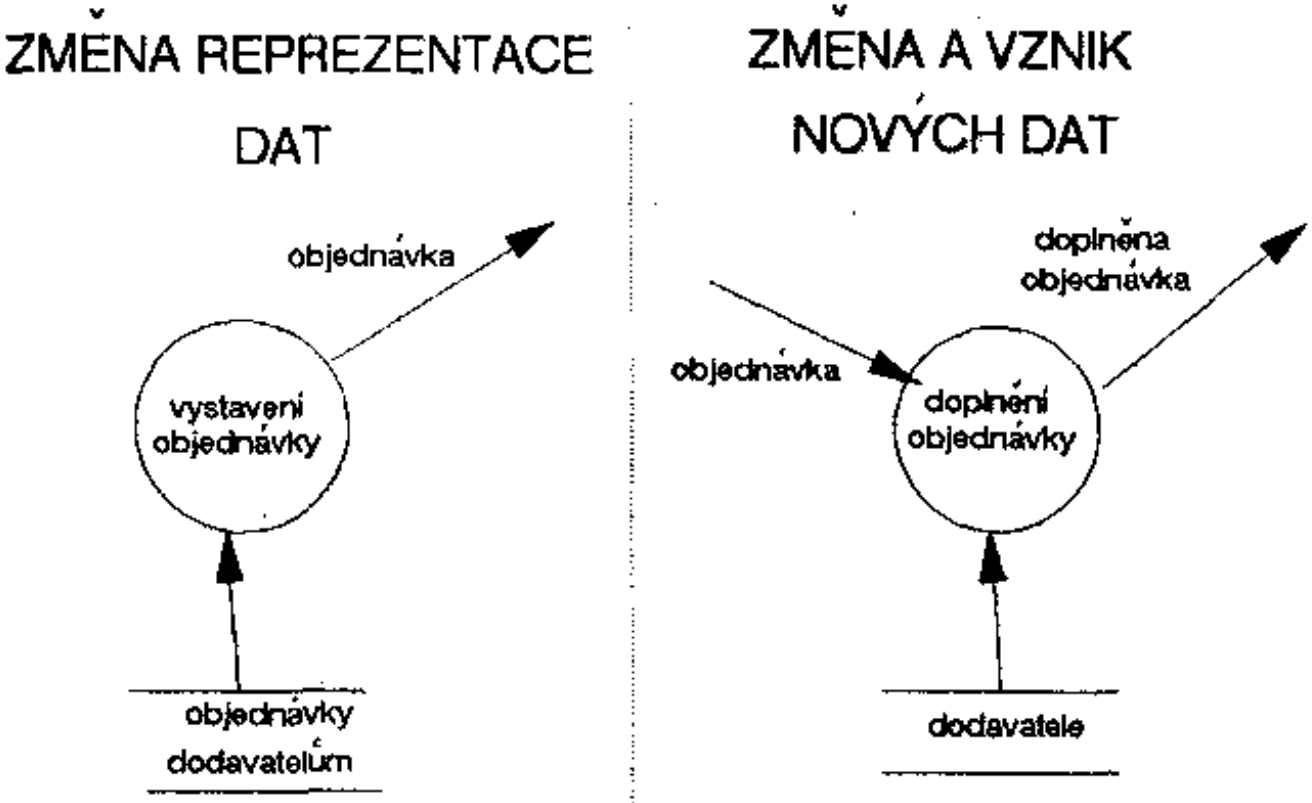
Rozlišujeme datové a řídící funkce. Nejprve budeme hovořit o prvním typu.

Datová funkce může vyjadřovat následující dva procesy:

a) fyzickou transformaci dat, tj. změnu reprezentace dat.

b) změnu stavu určité části dat, tj. změnu bodnot údajů, vznik nových údajů.

Druhá situace nastane, např. jestliže data vstupují do "ověřovacího" procesu a opouštějí tento proces jako "ověřená data". Prohlédněte si obr. 3.



Obr. 3. DFD - Dva typy datových informací

Aby bylo jasné, co který proces vyjadřuje, musí mít **název.**

Pro přehlednost DFD je výhodné, jestliže lze z označení procesu (funkce) rozpoznat, kam patří. K tomu se nejčastěji používá "desetinného třídění". Každá funkce má potom kromě svého názvu ještě jednoznačné číslo. Toto číslo se skládá z čísla bezprostředně nadřízené funkce, do jejíhož rozkladu označovaná funkce patří (např. 3.1.2), a přiděleného čísla v rámci úrovně (např. 5, celé označení funkce: 3.1.2.5.). Pozor, že čísla v rámci úrovně jsou identifikační a nevyjadřují pořadí provádění.

DATOVÝ TOK

vyjadřuje přesun dat/informací z jedné části systému do jiné, nebo z okolí systému do systému nebo ze systému do okolí. Znázorňuje se šipkou (data "tečou" naznačeným směrem, je možné použít i dialog flow - stejná data tečou oběma směry).

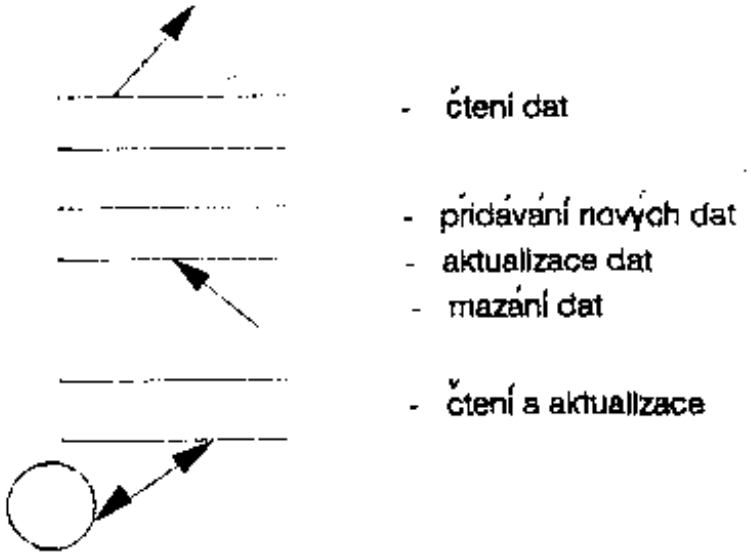
Datový tok musí mít známý obsah a musí být pojmenovaný (s výjimkou datových toků do storu a ze storu - viz dále). Název datového toku musí daná data reprezentovat a jasně vyjadřovat jejich obsah. (Př. vhodného názvu datového toku: "objednávka", nevhodného: "data", "růžový formulář").

Datové toky neobsahují žádná řídící data, ale pouze ta data, která jsou systémem zpracovávána nebo propouštěna. U programových systémů jsou obsahem datových toků zprávy, čísla, znaky, záznamy, bity, u jiných systémů to může být i vyjádření dat spolu s materiálním tokem (některé DFD rozlišují i materiální toky - viz lit. /Page-Jones/).

Možnosti propojení procesů datovými toky jsou dány obecnou syntaxí DFD a často i příslušným automatizovaným nástrojem pro kreslení DFD (CASE systémem).

DATA STORE

je schématicky znázorněn na obr. 4. Vyjadřuje "depozitář" dat (data uchovaná pro jejich pozdější použití). Znázorňuje se pomocí dvou rovnoběžek, mezi nimiž je umístěn jeho **název.**



Obr. 4. DFD - Data store pro data uchovávaná v systému

Význam data storu je tedy "místo dočasného uchování dat", což není to samé jako "soubor". Může být implementován různě (jako pole, soubor - obyčejný nebo databázový, ale i jako cokoliv jiného, například šanon, kniha apod.). Používá se všude tam, kde mezi procesy existuje časově zpožděné předávání dat (asynchronní). Důvody, proč si dva procesy nemohou předat data přímo, bez uchovávání v data storu, jsou různé: oba procesy neběží současně, nebo každý proces běží na jiném hardwaru, nebo to tak uživatel chce...

Název data storu je většinou v množném čísle. (Př. názvu data storu "dodavatelé", nevhodný název: "šuplík", "soubor", "databáze").

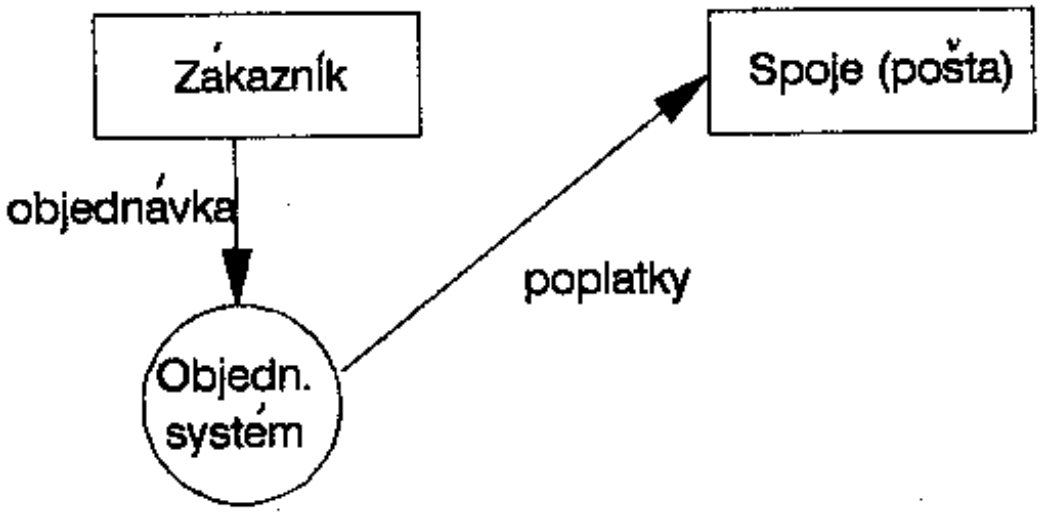
Pro každý store musí existovat datový tok do storu i ze storu! Tento tok může vyjadřovat:

- 1 skupinu (výskyt) dat (např. zaměstnanec),
- víc skupin dat vybíraných ze storu,
- část jednoho výskytu (např. jen telefon),
- část z více výskytů.

Data store je pasivní - data do a ze storu musí vždy téci přes transformaci dat.

TERMINÁTOR

je uveden na obr. 5. Znázorňuje externí zdroj nebo místo určení dat (někdy se též nazývá externí entita - objekt).



Obr. 5. DFD - Terminátory

Vyjadřuje tedy objekt v okolí systému, s nímž systém komunikuje. Může to být člověk, skupina lidí (oddělení), skupina oddělení ve stejné organizaci/podniku, ale vždy vně modelovaného systému. Může to být i jiný systém.

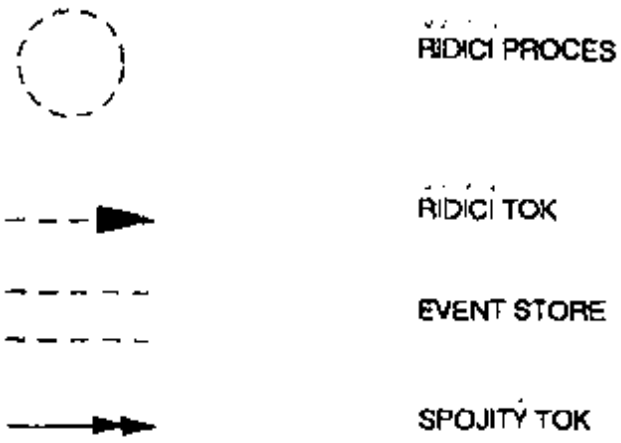
Graficky se značí čtvercem/obdélníkem. Vyjadřuje uživatele modelovaného systému nebo s kým je uživatel ve spojení (oddělení účetnictví, finanční odbor apod.).

Analytik a designér nemohou změnit kontext, ani způsob práce terminátoru, ani organizaci nebo interní postupy spojené s jeho činností. Terminátor je součástí okolí systému a jeho činnost nemůžeme ovlivnit.

I terminátor by měl mít výstižný název vyjadřující typ externího zdroje/místa určení (např. "zákazník", a ne "pan Krátký").

PROCES ŘÍDÍCÍ

je uveden na obr. 6. Označuje se přerušovaným kroužkem. Koordinuje a synchronizuje činnosti jiných procesů vyjádřených v DFD. Jeho vstupy a výstupy mohou být pouze řídící toky. Z důvodů přehlednosti (dobré strukturovanosti) smí být řídící proces v jednom DFD nejvýše jeden. Je popsán STD (viz kapitola 4.5). Vstup dostává tehdy, jestliže nastane nějaká událost (např. uživatel stiskne tlačítko na zapojení určitého přístroje) nebo některý proces skončí svoji činnost.



Obr. 6. DFD - Notace řídicích transformací

ŘÍDÍCÍ TOK

je představován přerušovanou šipkou. Znamená signál nebo přerušení. Řídící tok nese binární signál (on/off). Nemá datovou reprezentaci. Říká - "proveď", "počkej" (trigger, enable/disable).

Řídící proces nemusí být schopen okamžitě reagovat na signál nebo může více signálů přicházet naráz. V těchto případech je třeba signály uchovávat jako data, k čemuž se používá tzv. event store.

4.1.4. Hierarchie DFD

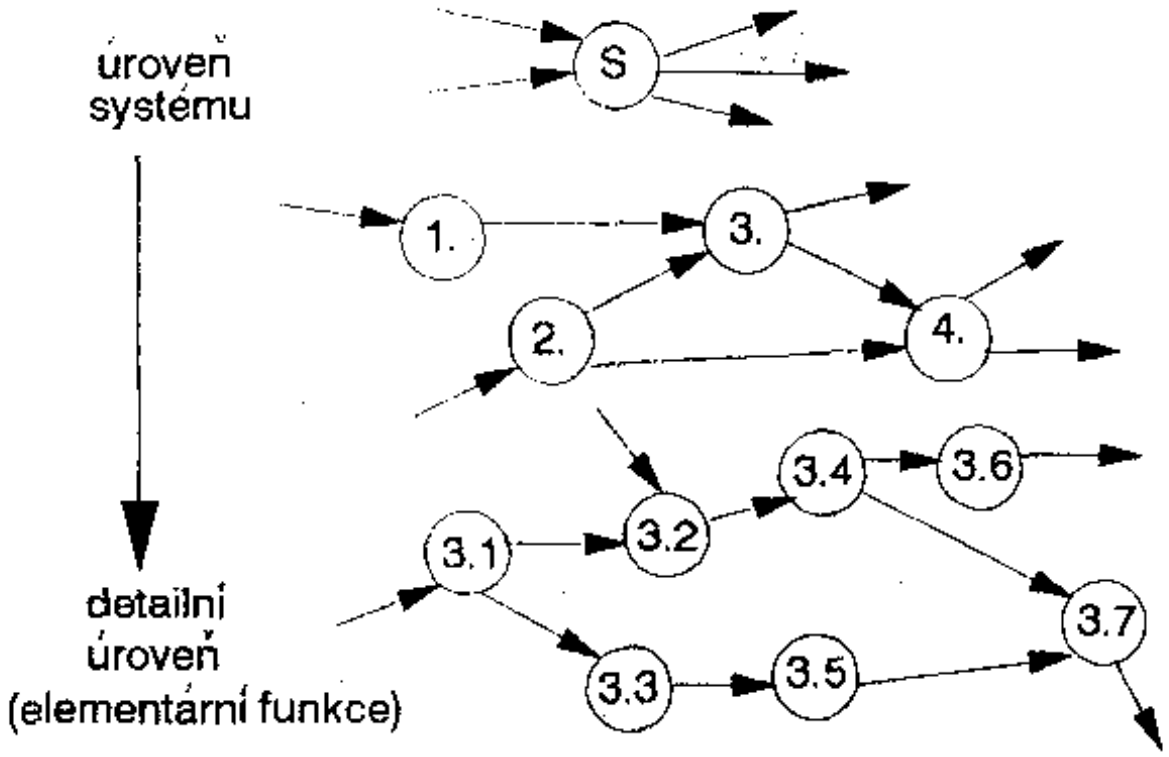
Model systému vyjádřený pomocí DFD diagramů má stromovou (hierarchickou) strukturu. Na základě podrobnosti rozkladu lze rozlišovat **DFD diagramy různých úrovní**. Pokud vyvíjíme informační systém podle principů strukturovaného přístupu (shora dolů), začínáme od přehledového diagramu a pokračujeme ke stále detailnějším diagramům. Tak vznikne hierarchie DFD diagramů. V této hierarchii lze rozlišit úrovně

- vrchní (top),
- střední (middle),

- spodní (bottom).

Hierarchie DFD je schematicky znázorněna na obr. 7. Na vrcholu hierarchie je pouze jeden DFD, tzv. **kontextový diagram** (obsahuje celý systém jako jednu funkci). Kontextový diagram reprezentuje hranice systému a všechny zdroje a místa určení (terminátory) dat.

Pokračujeme-li vnitřní strukturou této jedné funkce, dostaneme se na následující nižší úroveň (bezprostředně podřízenou kontextovému diagramu) - **DFD úrovně 0.** Obsahuje základní funkce systému (případně subsystémy) a jejich vztahy vyjádřené prostřednictvím datových toků a data storů).



Obr. 7. DFD - Hierarchie rozkladu a číslování procesů

Stejným způsobem pokračujeme v analýze funkcí této nižší úrovně. Nejnižší úroveň obsahuje **elementární funkce**, též označované jako primitivní funkce. Jsou to takové funkce, které je ještě schopen rozpoznat uživatel. Pod úrovní elementárních funkcí jsou již operace jako sčítání, odčítání, zápis dat apod. Elementární funkce je tedy souhrnem činností zpracovávajících data, pro který platí:

- činnosti se provádějí jako celek
- jsou buď všechny ruční, nebo automatizované,
- jsou opakovatelné,
- jsou elementární uživatelskou funkcí.

4.1.5. Pravidla tvorby DFD

Číslování procesů

Pravidla číslování procesů jsme již uvedli.

Shrnutí:

Vychází se z DFD úrovně 0. Všechny funkce pod touto nultou úrovní mají své pořadové číslo (pozor! nejedná se o pořadí provádění). Pokud stejným způsobem (rozkladem jedné funkce) získáme další podúroveň, přidáme ke všem takto vzniklým funkcím opět pořadové číslo - přidáním k pořadovému číslu funkce, z níž daný rozklad vznikl. Takové číslování identifikuje jednak úroveň rozkladu, do níž funkce patří, a jednak i proces, jehož je daná funkce rozkladem. Tedy číslování se provádí shora dolů po úrovních a v rámci jedné úrovně.

Názvy procesů (funkcí)

Pojmenování procesů je dost těžké - jméno by mělo být stručné, výstižné, má vyjadřovat funkci (co se děje). Např. "příjem pacienta", "vystavení faktury", "ověření telefonního čísla". Naprosto nevhodné názvy jsou takové jako "zpracuj data" nebo "zajímej se o zákazníka". Pro označování procesů se má používat slovníku uživatele, ale pozor na interní žargon. Názvům by měli rozumět i uživatelé v obdobných organizacích. Nepoužívat názvů jako subsystém, procedura, funkce apod.

Složitost DFD

Nevytvářejte příliš komplexní DFD. DFD musí být pochopitelný pro uživatele, analytika, designéra. Nevytvářejte DFD s velkým množstvím procesů a toků. Jediný DFD, který se nesmí zjednodušovat, je kontextový DFD.

Jeden DFD by neměl být větší než 1 stránka formátu A4. Jeden DFD by neměl mít více než 9 funkcí. Pokud obsahuje pouze 2 funkce, je třeba ověřit jejich opodstatněnost (není chybně rozklad vyšší úrovně?). Platí zde "magické číslo 7 ± 2".

Přehledné a esteticky uspořádané DFD

DFD musí být technicky správný (přesný, konzistentní), přijatelný a srozumitelný pro uživatele, úhledně nakreslený (slouží k prezentaci řešení).

Vnitřní konzistence DFD a konzistence se souvisejícími DFD

Konzistenci DFD zajistíme dodržováním všech již zmíněných a některých dalších zásad, které nyní stručně shrneme, příp. vysvětlíme.

Při kontrole konzistence hierarchické dekompozice do úrovní prověříme, zda:

- horní (top) - má jen "podúroveň",
- středová (middle) - má "nadúroveň i podúroveň",
- spodní (bottom) - má pouze "nadúroveň", neboť obsahuje již elementární funkce)

Každý DFD diagram musí mít své číslo (identifikaci procesu, jehož je rozkladem) a vysvětlující záhlaví.

Důležitá jsou pravidla týkající se datových toků:

a) nesmí existovat transformace (proces, funkce), která "bez pomoci" vstupů produkuje data. Tento "samogenerující" proces nemá žádné vstupní datové toky a má jen výstupní;

b) nesmí existovat proces, který pouze spotřebovává data. Tato "černá díra" má jen vstupní datové toky a žádné výstupní;

c) data story smějí být propojeny jen pomocí funkce. To znamená, že datový tok do/ze storu musí vycházet z procesu, nebo do něj ústit;

d) datový tok z/do terminátoru musí vždy jít přes proces.

e) datové toky mezi funkcemi znázorňují pouze předávaná data. Nevyjadřují volání jedné funkce druhou, ani předávání řízení;

f) datový tok s týmž názvem může být pochopitelně použit v DFD na více místech. Znamená to však, že se skutečně jedná stále o týž datový tok, tok se stejným obsahem.
Konzistence mezi úrovněmi v hierarchii DFD se dosahuje dodržováním dvou zásad:
   - Musí být zachováno číslování funkcí, označování (a typy toků) nadřízeného a podřízeného DFD.
   - Všechny vazby vyjádřené v úrovni n musí být též vyjádřeny na úrovni n-1.

   Netýká se to:
   - triviálních chybových toků,
   - toků ze a do storů, které jsou v rámci dekomponované funkce lokální,
   - toků mezi podfunkcemi.

   Pro funkce platí:
   - neznázorňují se žádné inicializační ani závěrečné procedury (start/stop procedury)

- neznázorňují se cykly mezi funkcemi
- neřídící (datové) procesy nesmějí být propojeny řídícími toky
- nelze mít dvě funkce se stejným názvem

Pro data story doporučujeme dodržovat tyto zásady:

a) v DFD se data story objeví až na té úrovni, kde jsou viditelné funkce, které do storů píší a ze storů čtou;

b) vyhledání pro aktualizaci data storu se chápe jako součást zápisu do data storu, nevyznačuje se zvláštní šipkou. Šipka směrem dovnitř znamená jakékoli provádění změn (přidávání dat, aktualizace, rušení);

c) v data storu jsou uloženy výskyty dat se stejnou strukturou. Jestliže tok ze storu (do storu) přenáší celý výskyt, nemusí být tento tok pojmenován. Datový tok je v tomto případě určen obsahem a názvem storu. Doporučujeme dodržovat následující pravidla pro názvy datových toků ze/do storu:
   - datový tok není označen ⇒ jedná se o jeden výskyt;
   - datový tok je označen názvem storu ⇒ jedná se o jeden nebo více výskytů;
   - datový tok má své jedinečné jméno ⇒ jedná se o část jednoho nebo více výskytů.

Přesný obsah datového toku je zřejmý z minispecifikace procesu a popisu data storu ve slovníku dat. Nezapomínejte, že tedy toky ze storu a do storu mohou vždy obsahovat pouze ta data, která ve storu jsou či mohou být. Z faktur objednávku tvořiti nelze.

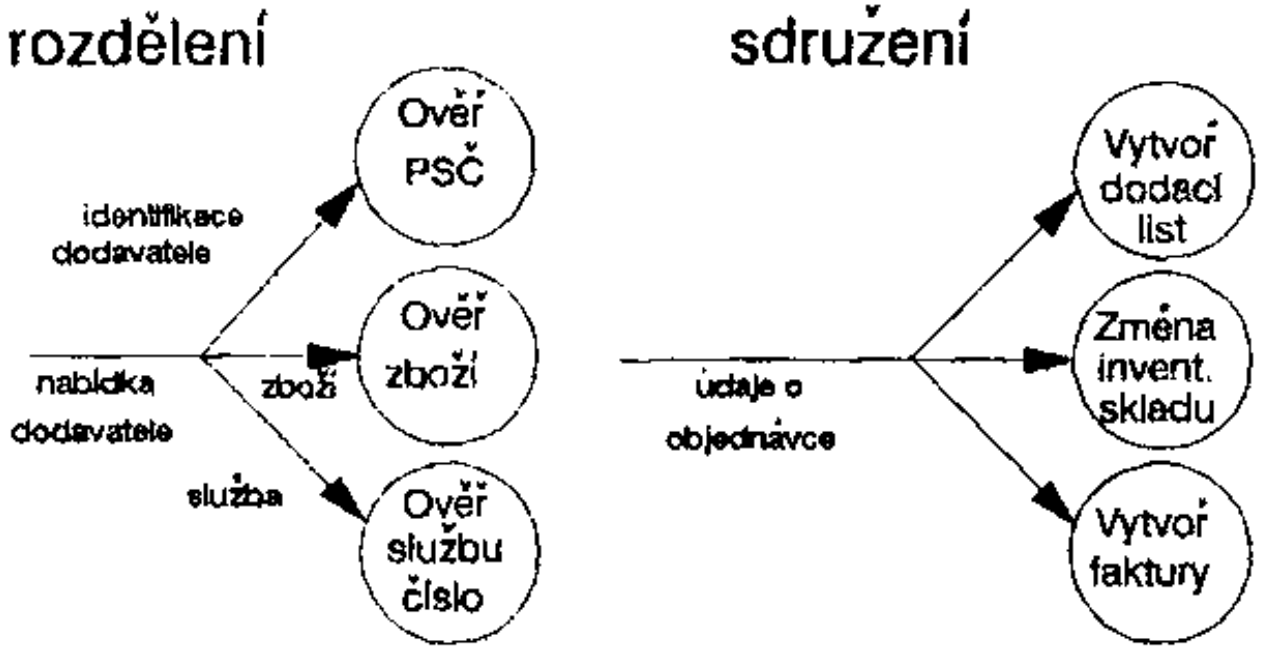
Kontextový diagram znázorňuje pouze vstupy a výstupy **celého systému** = jedné funkce. Jestliže nejsme schopni tuto funkci pojmenovat, provedli jsme první ověřené soudržnosti systému. S negativním výsledkem.

DFD obsahují esenciální procesy jak ručně prováděné, tak automatizované.

Konzistenci kontrolují systémy CASE (každý z nich kontroluje vybraná pravidla, případně může přidávat i svá vlastní).

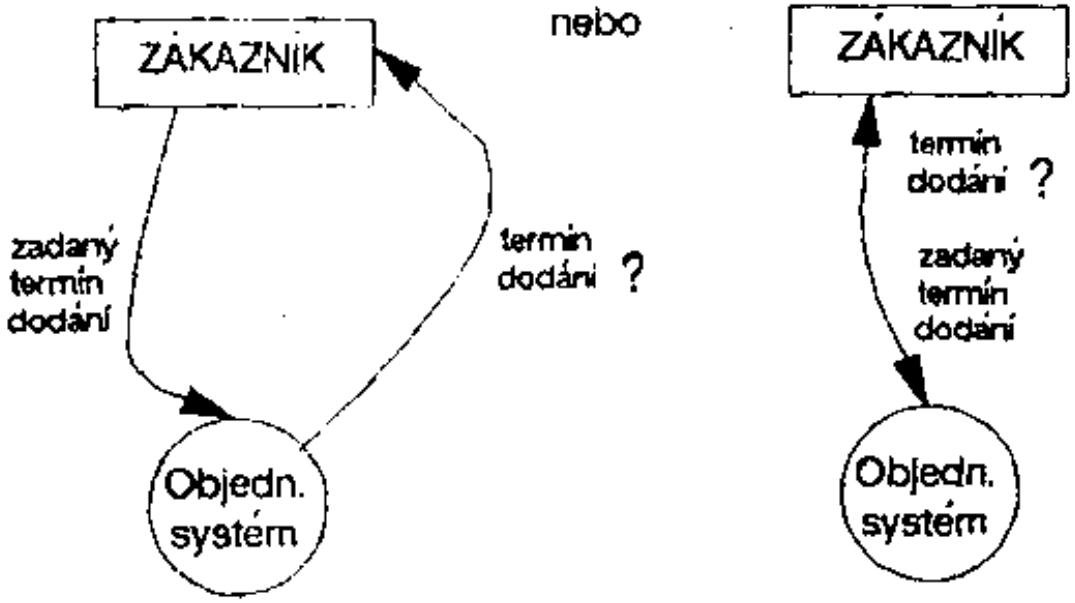
Další možnosti zápisu datových toků

Datové toky se mohou dle potřeby rozdělovat, nebo sdružovat. Ilustrativní příklad je uveden na obr. 8.



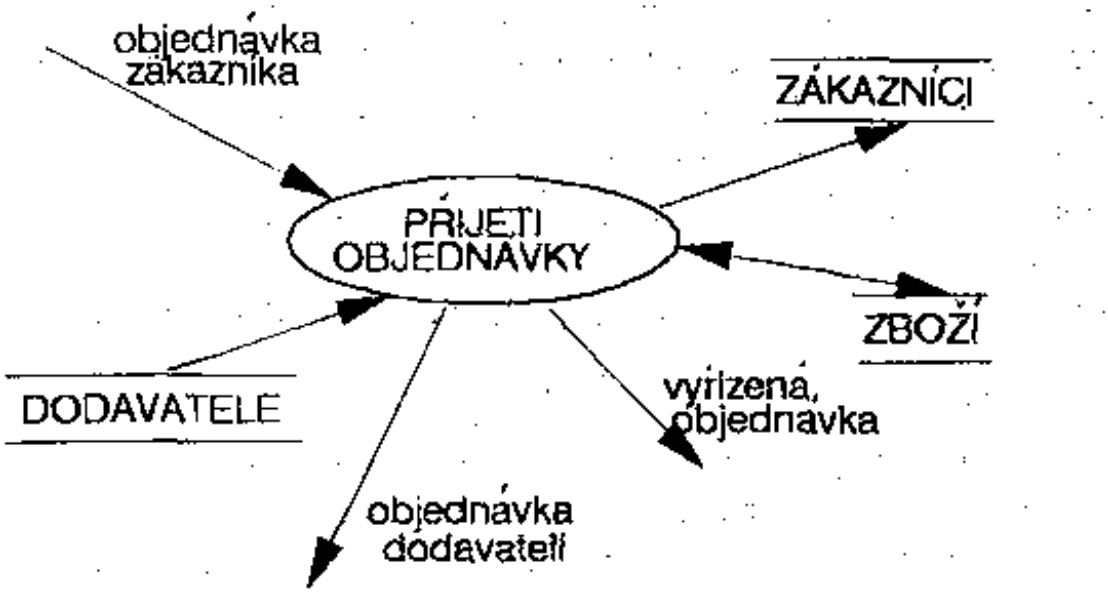
Obr. 8. DFD - Příklad rozdělení a sdružení datového toku

Datový tok může vyjadřovat dialog (nejčastěji mezi "hraniční" funkcí a terminátorem). V tomto případě použijte jeden ze dvou způsobů zápisu zachycený na obr. 9.



Obr. 9. DFD - Vyjádření datového toku typu dialog

Jiným případem možného použití oboustranné šipky je čtení i aktualizace dat v data store jednou funkcí. Tato situace nastává tehdy, jestliže funkce něco přidává do data store (nebo mění v data storu) a zároveň používá data ze storu k vytvoření svého výstupu. Příklad ilustrující notaci čtení i aktualizaci dat jednou funkcí je na obr. 10.



Obr. 10. DFD - Příklad současného čtené i aktualizace dat jednou funkcí

4.1.6. Otázky spojené s úrovněmi DFD.

Kolik úrovní DFD?

Každá úroveň by neměla mít víc než 9 procesů a odpovídajících data storů. Jsme-li schopni napsat rozumnou minispecifikaci pro každý proces na 1 stránku A4, skončíme s rozkladem úrovní DFD. Jestliže je proces stále ještě moc složitý, pokračujeme v dekompozici.

Existuje doporučení o správném počtu úrovní v typickém systému?

U jednoduchých systémů 2-3 úrovně. Střední systémy 3-6 úrovní. Velké systémy 5-8 úrovní.

Poznámka: Pokud by bylo na každé úrovni 7 procesů, potom na třetí úrovni docílíme počtu 343 funkcí, při pěti úrovních 16 807 funkcí a při devíti úrovních 40 353 607. Není to málo.

Musí být všechny části systému rozděleny do stejné úrovně?

NE! Některé části systému jsou komplexnější, jiné méně. Ale pokud mám například funkci 2 rovnou elementární a v rozkladu funkce 3 existu-

je funkce s číslem 3.2.1.1.3.5 - měl by tento fakt napovědět, že je nejspíš nějaká chyba v návrhu.

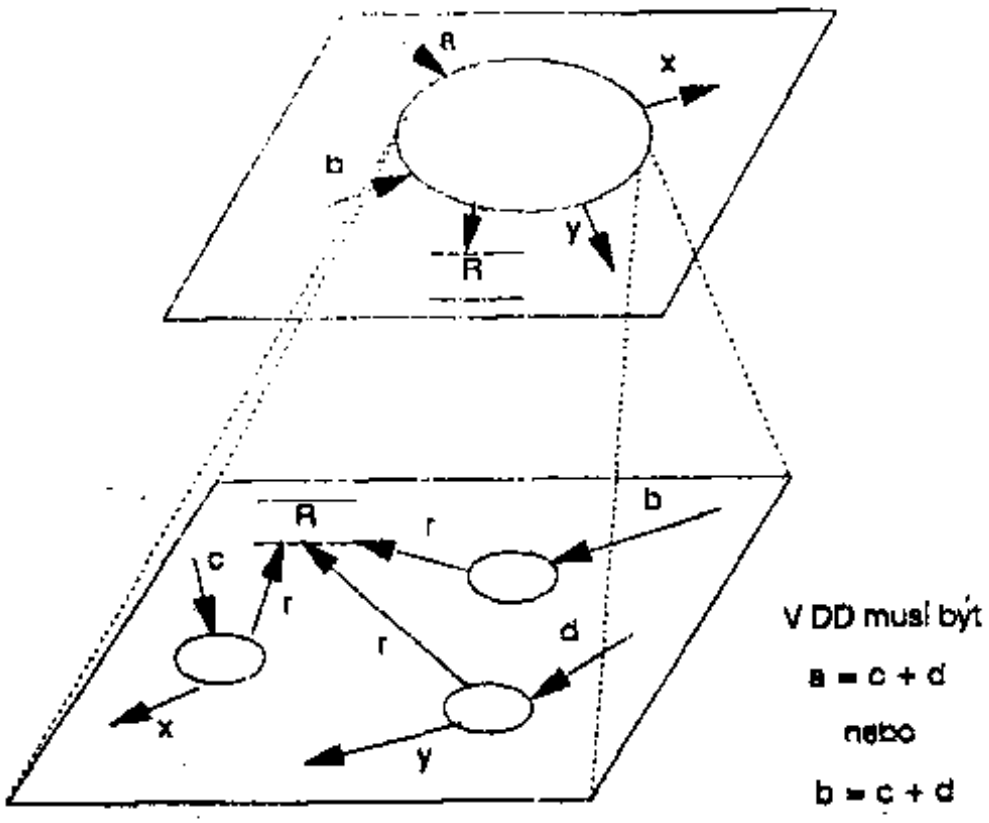
Jak prezentovat úrovně uživateli?

Záleží na tom, jak komu. Uživatelé se liší postavením v organizační struktuře, ochotou a schopností komunikovat, ochotou a schopností pochopit řešený problém. Řediteli je obvykle vhodné prezentovat kontextový diagram a DFD úrovně 0. Obsluhu systému zase naopak zajímají nejspodnější úrovně té části systému, která se jich týká. Obecně platí, že se DFD vždy prezentují shora dolů. DFD doplňujeme s potřebnou další dokumentací a výkladem, aby se zajistila srozumitelnost uživatelům.

Prezentace DFD nemusí korespondovat se strategií jejich vytváření.

Jak ověřit konzistenci úrovní DFD?

Tato činnost je bezpodmínečně nutná, protože různé úrovně jsou většinou vyvíjeny různými lidmi. Konzistence DFD se vždy kontroluje s bezprostředně vyšší úrovní DFD - datové toky, které vstupují/vystupují do/z funkce na jedné úrovni musí korespondovat se vstupy a výstupy nižší úrovně (rozkladem dané funkce). Složené toky jsou popsané v DD. Ilustrativní příklad je na obr. 11.



Obr. 11. DFD - Hierarchický rozklad musí být konzistentní

Co s data story na jednotlivých úrovních?

Data store je třeba zobrazit až na té úrovni, kde zajišťuje rozhraní mezi dvěma nebo více procesy. Potom již musí být tento store zobrazen ve všech nižších úrovních rozkladu funkce. Alespoň na nejvyšší úrovni, kde se store poprvé vyskytuje, musí mít vstupní i výstupní datový tok.

Jak skutečně provádět rozdělování?

Nemusí být vždy pravda, že se DFD musí navrhovat shora dolů. Existují různé postupy, jak k hierarchii DFD dospět:

- rozdělováním procesů (funkční dekompozice),
- postupem podle datových toků,
- na základě analýzy událostí (Yourdon 1989).

Podrobněji viz kapitola 5.

4.2. Diagram entit a vztahů - ERD (Entity Relationship Diagram)

DIAGRAM ENTIT A VZTAHŮ

Diagram entit a vztahů je grafický nástroj, který se používá k vyjádření datového modelu (datových objektů, které jsou v systému uchovávány a vztahů mezi nimi).

Data systému jsou stejně důležitá jako procesy (funkce), které v systému probíhají. Ve funkčním modelu jsou také zobrazena uchovávaná data, ale chybí v něm důležité informace, a to informace o vzájemných vztazích mezi daty. Struktury dat a jejich vztahy mohou být tak složité, že je nutné je zprůhlednit a ověřit nezávisle na funkční struktuře (tedy na struktuře jejich zpracování). Navíc určití uživatelé (většinou ti na úrovni vrcholového vedení a vyšších vedoucích - jako je ředitel, náměstci, vedoucí oddělení) se nezajímají ani tak o běžné denní operace, jako spíš o data (jaká data potřebují k řízení a pro fungování podniku, kdo má za která data odpovědnost, kdo k nim má mít přístup apod.).

Datový model je dobrým nástrojem pro komunikaci s pracovníky oddělení správy dat zkoumané organizace, a to jak při analýze požadavků, tak při modelování nového systému.

Dále je ERD také vhodným prostředkem při komunikaci s pracovníky oddělení správy databáze. Ti budou většinou nově navržená data pro auto-

matizované zpracování implementovat. Zajímají je vztahy mezi daty, práva přístupu, rozsahy dat, integritní omezení.

Pro analytika je ERD důležitý hlavně pro zobrazení vztahů mezi data story z DFD. V DFD lze vztahy mezi daty modelovat pouze pouze ve specifikaci procesů, tedy verbálně, a proto méně transparentně. Navíc ERD umožňuje vyjádřit i takové vlastnosti dat, jako je specializace/generalizace.

Dokonce v extrémních případech lze modelovat systémy, které neprovádějí "žádné procesy", pouze uchovávají a vyhledávají velké objemy dat - tj. dotazovací systémy nebo SYSTÉMY NA PODPORU ROZHODOVÁNÍ. Při analýze a návrhu těchto systémů se pozornost věnuje hlavně datovému modelování. Také tím, že nejdříve vytvoříme datový model, často zjistíme, jaké jsou potřeba funkce.

Výsledkem datového modelování je tedy **datový model**, který odráží (modeluje) organizaci nebo její část z hlediska potřebných dat a vztahů mezi nimi.

První techniky datového modelování vznikly již v 70. letech jako důsledek řady problémů při údržbě systémů, které byly způsobeny hlavně chybami v datech (duplicity a nekonzistence). Velké pozornosti se jim dostalo hlavně s vývojem databázových systémů.

Existuje několik notací grafického vyjádření datového modelu (Chen, Martin, Bachman ...). Yourdon používá svoji notaci odvozenou z Flavina. Většina CASE prostředků používá jednodušší notaci, vycházející z Martinovy notace.

Prvky ERD (význam a způsob kreslení)

ENTITA

Entita je jakýkoliv objekt, který je předmětem zájmu při z hlediska analyzovaného systému, o kterém chce organizace uchovávat data. Představuje konkrétní věci, osoby, předměty (zboží, zákazník, budova, fakulta), události (pronájem bytu, dopravní nehoda, složení zkoušky), nebo pojmy (norma materiálu na operaci, zásoba zboží na skladě).

Entita vždy vyjadřuje typ daného předmětu zájmu a většinou má více výskytů. Název entity je tedy v jednotném čísle, protože pojmenovává typ (společné vlastnosti předmětu zájmu).

Identifikátor entity musí být:

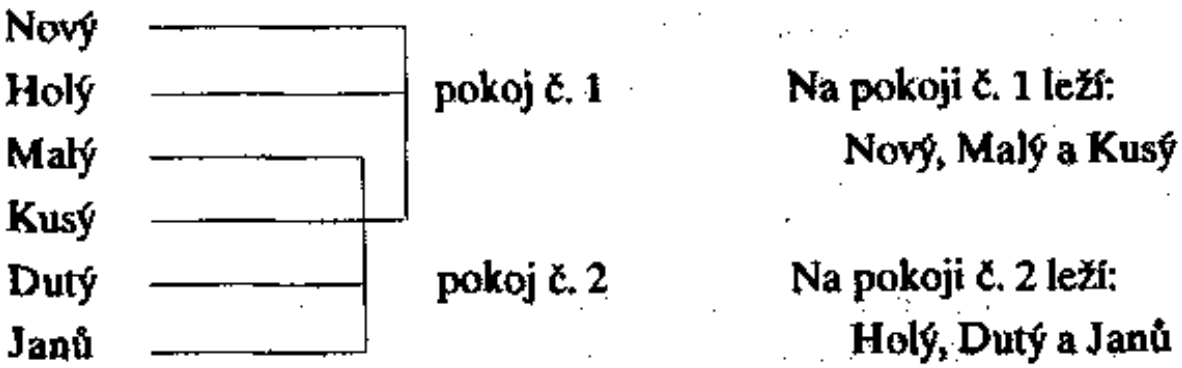
- jedinečný (jedna hodnota identifikátoru jednoznačně určuje právě jeden výskyt entity),

- úplný (pro každý výskyt entity existuje právě jeden identifikátor),
- minimální (neobsahuje žádné atributy navíc, v identifikátoru složeném z více atributů, nelze nalézt takovou podmnožinu, která by byla jedinečná a úplná).

Příklad: Entita POLOŽKA OBJEDNÁVKY je identifikována atributy číslo objednávky a číslo zboží.
Zápis ve slovníku dat:
POLOŽKA OBJEDNÁVKY = číslo objednávky + číslo zboží + objednané množství zboží

VZTAH

Vztah mezi entitami vyjadřuje ve skutečnosti množinu vztahů mezi objekty (jedná se o vztahy mezi výskyty entit). Každý výskyt vztahu vyjadřuje spojení mezi nula, jedním nebo více výskyty jedné entity se žádným, jedním nebo více výskyty jiné entity.



Vztah může spojovat 2 i více výskytů stejné entity.

Vztah vyjadřuje informaci, kterou si systém musí pamatovat, informaci, která nemůže být vypočtena nebo odvozena z informací jiných. Např. pokud by neexistoval vztah mezi objekty PACIENT a POKOJ, nebylo by možné zjistit, který pacient leží na kterém pokoji.

Příklad: DODAVATEL je typ objektu
Triola a.s. je výskyt entity DODAVATEL
Tiba s.p. je další výskyt entity dodavatel.

Entita se graficky většinou vyjadřuje obdélníkem.

ATRIBUT

Atribut je elementární datový prvek, který entitu blíže charakterizuje, popisuje.

Příklad: entita: *PACIENT*
má atributy: *jméno pacienta*
adresa pacienta
rodné číslo pacienta
váha pacienta

Ke každé entitě se vytvoří seznam všech jejích atributů.

Atributy mohou mít vzhledem k entitě různou kardinalitu:

- k jednomu výskytu entity jeden výskyt atributu;
- nebo **opakovaný atribut** - k jednomu výskytu entity několik výskytů atributu (např. KNIHA má maximálně 3 autory, autor knihy je opakovaný atribut).

Dále mají atributy různou volitelnost:

- **totální atribut** - pokud existuje výskyt entity, má takový atribut hodnotu (např. každý PACIENT je nějak vysoký),
- **parciální atribut** - pokud existuje výskyt entity, nemusí mít daný atribut hodnotu (např. plešatý PACIENT nemá hodnotu atributu BARVA VLASŮ).

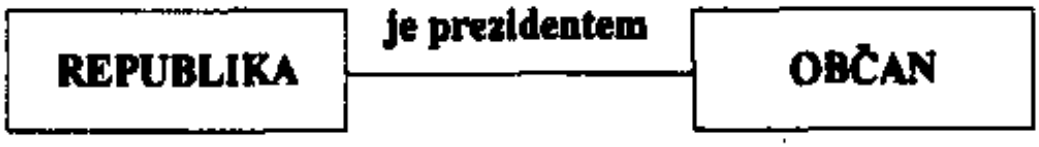
Každá entita musí mít **identifikátor** - množina atributů dané entity a atributů entit "nadřízených" (entit, ke kterým má daná entita vztah) - tzv. externích identifikátorů.

Mezi dvěma stejnými entitami může existovat více různých vztahů. Proto i vztah musí mít svůj název. Např. učitel může garantovat určitý předmět a zároveň může učit i předměty, které negarantuje - mezi entitami UČITEL a PŘEDMĚT existují vztahy "učí" a "garantuje".

Vztah mezi dvěma entitami se nazývá binární vztah. Některé notace ERD diagramu (například Chenova) umožňují vyjádřit i vztahy mezi více než dvěma entitami (vícenásobné vztahy).

U ostatních notací je nutné vícenásobné vztahy vyjádřit "umělou" entitou.

V Martinově notaci se vztah vyjadřuje čárou mezi entitami.

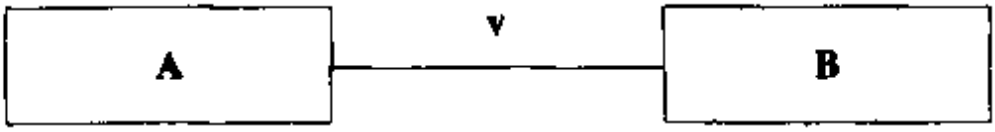


Každý nalezený vztah se pojmenuje se (nejraději pomocí sloves) a dále se analyzuje .

Kardinalita vztahu

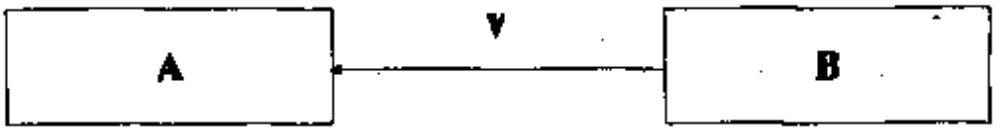
Stejně jako atributy mohou mít vztahy mezi entitami různou **kardinalitu** (vyjadřuje mocnost vztahu mezi výskyty entit).

Vztah 1:1 znamená, že k jednomu výskytu ENTITY A existuje právě jeden výskyt ENTITY B a naopak.



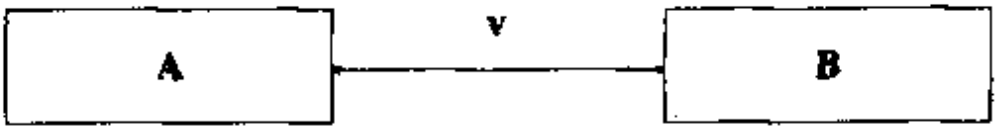
Příklad: jedna republika má právě jednoho prezideta a jeden prezident je prezidentem právě jedné republiky

Vztah 1:N znamená, že k jednomu výskytu ENTITY A existuje více výskytů ENTITY B a k jednomu výs kytu ENTITY B exituje právě jeden výskyt ENTITY A.



Příklad: učitel garantuje více předmětů a předmět je garantován právě jedním učitelem

Vztah N:M vyjadřuje, že k jednomu výskytu A existuje více výskytů B a naopak.



Příklad: na jedné objednávce je uvedeno více druhů zboží a jeden druh zboží je objednán na více objednávkách.

Volitelnost vztahu (parcialita/totalita)

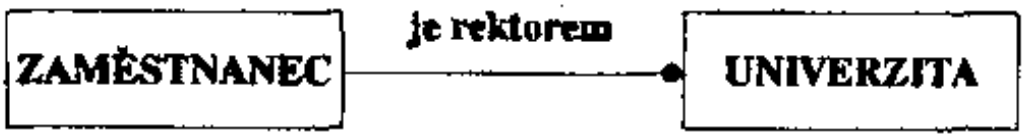
Vztah mezi entitami je **parciální**, pokud k jednomu výskytu entity A může a nemusí existovat výskyt entity B. Tj. vznikne-li nový výskyt entity A, nemusí vzniknout ani jeden výskyt vztahu k entitě B. Parcialita se vyjadřuje

buď puntíkem na té straně vztahu, kde nemusí existovat výskyt entity, nebo přerušovanou čárou apod.

Vztah mezi entitami je **totální**, pokud k jednomu výskytu entity A musí existovat alespoň jeden výskyt entity B. Pokud vznikne nový výskyt entity A, musí vzniknout i vztah k výskytu entity B.

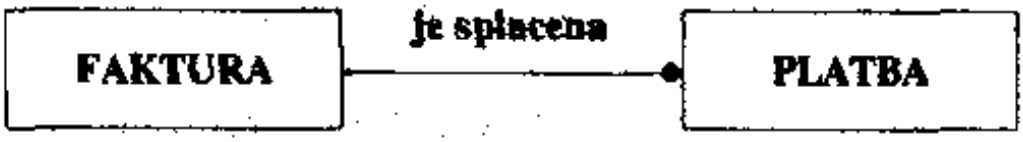
Příklady:

1.



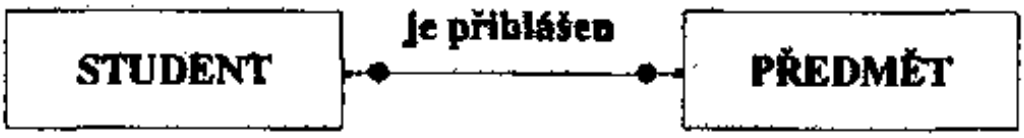
1 zaměstnanec může a nemusí být rektorem 1 univerzity.
Každá univerzita má právě 1 zaměstance jako rektora.

2.



1 faktura splácena žádnou nebo i více platbami
1 platba je k právě 1 faktuře

3.

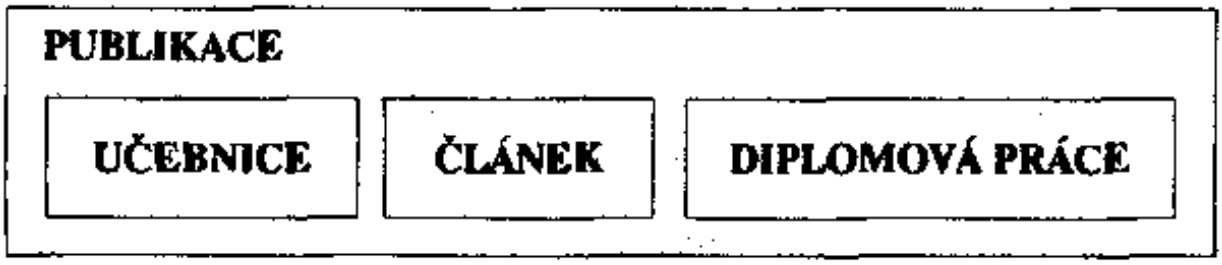


1 student může (ale nemusí) být přihlášen na více předmětů,
na 1 předmět může být přihlášen žádný, jeden nebo více studentů.

Generalizace/specializace

Některé typy objektů jsou složeny z obecného typu objektu (nadtypu) a jedné nebo více "podkategorií". Některé ERD mají možnost podtyp a nadtyp vyjadřovat.

Příklad:



$$PUBLIKACE = Číslo\ publikace + název\ publikace + {}_1\{autor\ publikace\}_2 + rok\ vydání + typ\ publikace$$

** Typ publikace je tzv. atribut specializa ce - určuje, o který podtyp se jedná **

*UČEBNICE = * má stejné atributy jako PUBLIKACE * + pořadí vydání + nakladatelství*

*ČLÁNEK = * má stejné atributy jako PUBLIKACE * + název časopisu + číslo časopisu + počáteční stránka článku*

*DIPLOMOVÁ PRÁCE = * má stejné atributy jako PUBLIKACE * + vedoucí diplomové práce + datum obhájení*

Podtypy mají stejné atributy jako nadtyp a navíc ještě mají své vlastní atributy. Podtyp může mít též vztahy k jiným entitám. Pokud má entita/nadtyp vztah k jiné entitě, potom tento vztah platí pro všechny podtypy.

Postup tvorby ERD

Při tvorbě datového modelu se vychází z reálného systému, který modelujeme (organizace, organizační struktura, data, která hrají důležitou roli).

Při tvorbě datového modelu se vychází hlavně:

- z toho, jak analytik pochopí modelovanou realitu (již existuje například DFD včetně popisu datových toků a data storů ve slovníku dat),
- z interview s uživateli,
- z ostatních možných zdrojů zjišťování dat (oddělení správy dat apod).

Stejně jako funkční model popsaný pomocí DFD, tak i ERD se většinou nepovede navrhnout napoprvé a musí se neustále upravovat a zpodrobňovat podle nově získávaných poznatků. Existují pravidla pro zpodrobnění datového modelu - vedou k nalezení (odvození) dalších entit a naopak k odstranění nadbytečných entit.

Nalezení nových entit

První datový model se vytvoří na základě prvního pochopení oblasti zájmu. Vytipují se entity a jejich vztahy. Pro každý objekt se zkoumá:

- Jaký je hlavní účel objektu?
- K čemu ještě slouží?
- Z čeho se skládá?

Příklady:

V OBCHODĚ se prodávají DRUHY ZBOŽÍ,
DRUH ZBOŽÍ se objednává u DODAVATELE,

u DODAVATELe nás zajímá IČO, ADRESA a BANKOVNI SPOJENÍ.

Postup tvorby datového modelu

1. Výběr nejdůležitějších objektů - entit (OBCHOD, DODAVA VATEL, DRUH ZBOŽÍ) a hledáním se vztahů mezi entitami.
2. Nakreslení entit a jejich vztahů do ERD diagramu. Definice kardinality vztahů a parciálních vztahů.
3. Přidání atributů k entitám, označní parciálních atributů a určení ideti-fikátorů entit.
4. Kompletace hierarchie - pro každý typ specializace/ /generalizace se hledá atribut specializace a zkoumá se, zda je specializace úplná, případně proč tomu tak není.
5. Odstranění tranzitivních vztahů (nepřímých, které lze odvodit ze vztahů jiných).
6. Zaznamenání omezujících podmínek (integritních omezení) k entitám, např. čtenář si nemůže vypůjčit více než 10 knih.

Při přidávání atributů entitám v hrubém datovém modelu mohou být odha-leny nové entity, a to tehdy, pokud:

- Zjistí se, že určitý datový prvek lze přiřadit jako atribut jen některým výskytům entity, ale jiným ne (jedná se pravděpodobně o podtyp obecnější entity).
- Nalezne se datový prvek, který může být přiřazen jako atribut všem výskytům dvou různých entit (jedná se zřejmě o dva podtypy obecnější entity).
- Zjistíme, že některé datové prvky popisují vztahy mezi en titami (datový prvek nelze přiřadit ani jedné entitě). Potom ze vztahu vznikne entita nová, která bude mít vztahy na entity původní (např. atribut objednané množství nepopisuje ani entitu OBJEDNÁVKA, ani entitu ZBOŽÍ, ale vztah mezi těmito entitami. Vznikne nová entita POLOŽKA OBJEDÁVKY).

Odstranění nadbytečných entit z datového modelu

Může dojít k následujícím případům:

- Entita má jako atributy pouze identifikátor (je třeba prověřit, zda identi-fikátor nelze přidat jako atribut k jiné entitě, která má s původní entitou vztah - např. Otec zaměstnance, u kterého sledujeme pouze jméno, přidá se jako atribut k zaměstnanci).

- Entita má jen jeden výskyt (je-li navíc jediným atributem této entity identifikátor, může se jednat o konstantu),
- V modelu jsou tranzitivní vztahy (odvoditelné ze vztahů z jiných vztahů, a tudíž vztahy nadbytečné. Zbytečné vztahy se odstraní.
  Např. dodavatel → objednávka a
  dodavatel → objednaný výrobek a
  objednaný výrobek → objednávka
  potom je pravděpodobně jeden ze vztahů
  dodavatel → objednávka
  dodavatel → objednaný výrobek
  nadbytečný).

Závěr k datovému modelování

Datový model je potřebným pohledem na statickou část systému - vyjadřuje vztahy mezi typy dat, aniž by se popisovaly funkce, pomocí nichž jsou tato data vytvářena, nebo které daná data používají.

Vývoj datového a funkčního modelu se vzájemně velmi těsně ovlivňují. Se kterým modelem se začne, závisí na tom,

čemu dává přednost analytik, nebo čemu dává přednost uživatel, nebo na typu systému. V určité fázi si postup vždy vyžádá tvorbu druhého modelu. Někdy se dokonce oba modely vytvářejí paralelně dvěma skupinami lidí ("funkčními" a "datovými" analytiky), kteří spolu vzájemně konzultují.

Dnešní systémy jsou většinou značně rozsáhlé a potřebují jak funkční, tak datový pohled, vzájemně provázané (viz pravidla ověřování konzistence mezi nástroji modelování).

4.3. Slovník dat (DD - Data Dictionary)

Slovník dat slouží ke slovnímu formalizovanému popisu dat systému (a to dat z pohledu uživatele: "business data").

Obsahuje:

- význam datových toků a storů z DFD,
- složení dat (rozdělení na prvky) v datových tocích (dekompozici neelementárních dat - jako např. adresa zákazníka, která může být rozdělena na elementárnější položky - jako např. město, psč,...),
- složení dat v data storech,
- specifikaci možných hodnot a jednotek elementárních dat ve storech a tocích,

- údaje o vztazích mezi entitami, které jsou součástí ERD.

Notace zápisu v DD obvykle vychází ze zjednodušené BNF:

ADRESA = [prijmeni + jmeno + (titul) + ulice |
postovni schranka] + mesto + (PSC) + (zeme)

Použité symboly:

=	skládá se z ("adresa se skládá z"),
+	a ("prijmeni a jmena a ..."),
[|]	buď a nebo,
{ }	opakování (iterace): jednou nebo vícekrát,
( )	nepovinnost: může se vyskytovat 1x nebo vůbec,
* *	poznámka,
@ nebo _	identifikátor (klíčová položka) ve storu (entitě).

Definice datového prvku se uvádí pomocí symbolu "=". Ten se čte jako "je definován jako" nebo "znamená", tedy:

A = B + C - A znamená B a C
- A se skládá z B a C
- A je definováno jako B a C

Aby datový prvek byl definován úplně, měla by definice zahrnovat:

- význam datového prvku v rámci uživatelské aplikace - většinou ve formě poznámky (nebo tak, jak si řešitel dohodne konvence - např. zvláštní položka popisu datového prvku v systémové encyklopedii CASE SDW),
- složení datového prvku, pokud je složen ze smysluplných elementárních prvků,
- hodnoty, kterých může elementární datový prvek nabývat, případně i se specifikací formátu (např. počtu desetinných míst).

Příklad:

V IS nemocnice jsou datové prvky výška a váha definovány:
*Výška = * výška pacienta při přijetí do nemocnice **
** jednotka: centimetry, rozsah 30- 250 **
*Váha = * váha pacienta při přijetí do nemocnice **
** jednotka: kilogramy, rozsah 1 - 200 **

Slovník dat obsahuje obvykle velké množství životně důležitých informací. Pro zvýšení jeho přehlednosti a vypovídací schopnosti doporučujeme dodržovat i několik dalších pravidel.

Doplňující pravidla pro popis dat použitých ve funkční specifikaci v DD:

- Pro všechna data popsat omezení vzhledem k integritě a bezpečnosti a aspekty jako čas, množství a kvalitu.
- Pro výstupní data popsat stupeň potřeby těchto dat (musí být, měla by být, bylo by dobré je mít).
- Pro každý datový tok, datový prvek a data store použít jednu stránku A4 (kvůli aktualizaci dokumentace) - při ručním vedení.
- Každé synonymum musí odkazovat na standardní název.
- Stránky popisu skládat podle abecedy (pro rychlejší orientaci).
- Elementární datové prvky, jejichž význam je jasný z názvu (mají stejný význam ve všech IS), není třeba blíže vysvětlovat, ale je dobré specifikovat jednotku a možný rozsah (výčet hodnot).
  *Příklad: Pohlaví = * * (znamená bez poznámky)*
  ** hodnoty: [M|Z] **

4.3.1 DD - typy datových prvků

Parciální (nepovinné) datové prvky

Nepovinné datové prvky mohou, ale nemusí být součástí složeného datového prvku.

Příklad:

adresa-zákazníka = (dodací-adresa) + (platební-adresa)

- jen dodací-adresa nebo
- jen platební-adresa nebo
- dodací-adresa a platební-adresa nebo
- ani dodací-adresa a ani platební-adresa

Poslední případ je dost nesmyslný.

Selekce datových prvků

Datový prvek se skládá z právě jednoho z možných alternativních výběrů.

Příklad:

adresa-zákazníka = [dodací-adresa | platební-adresa | dodací-adresa + platební-adresa]

adresa zákazníka musí obsahovat buď

- dodací-adresu nebo
- platební-adresu nebo
- obě adresy.

Iterace datových prvků

Používá se k označení opakujících se částí datového prvku. Znamená *nula nebo více výskytů ...*

Příklad:

objednávka = název-zákazníka + adresa-zákazníka
+ {položka-objednávky}

- objednávka musí vždy obsahovat datové prvky název-zákazníka a adresa-zákazníka a též obsahuje nula nebo víc položek -objednávky. Tedy můžou být objednávky jen s jednou objednanou položkou nebo i s 200 položkami.
- uživatel může často určit dolní a horní hranici počtu opakování:
  a = 1{b}
  a = {b}15
  a = 1{b}15
  a = {b}

Alias (synonyma)

Alias znamená alternativní název pro datový prvek. Uživatelé často nazývají stejné věci různě - např. na Moravě a v Čechách, nebo v různých podnicích klient je to samé jako zákazník apod.

Příklad:

klient =
** alias zákazník **

Složení datového prvku je **popsáno pouze** u primárního názvu datového prvku, tedy u zákazníka - z důvodu minimalizace redundance! Přesto je lepší se synonymům vyhýbat, protože z názvu datového prvku není jasné, že jde o synonymum. Je lepší přesvědčit uživatele než si zkazit slovník.

4.3.2 Prověření úplnosti slovníku dat

Nejdříve analytik ověří, zda je DD kompletní, konzistentní a nerozporný. Potom jej ověřuje s uživatelem, ale vždy společně s další částí dokumentace - s DFD nebo ERD nebo STD.

Co analytik kontroluje:

- Zda má každý datový tok v DFD definici v DD.
- Zda jsou definovány všechny složené datové prvky.
- Nejsou-li některé datové prvky definovány vícekrát.
- Byla-li pro všechny definované datové prvky použita správná notace.

- Nejsou-li v DD nějaká data, která nejsou použita ani v DFD, ani v ERD, ani v STD.

4.3.3 Implementace DD

Ve středně velkých a velkých systémech může DD běžně obsahovat statisíce datových prvků. I malý systém má stovky datových prvků. Ruční vedení DD by bylo velmi obtížné, hlavně pokud se týká kontroly konzistence a redundance. Proto je dobré pokud možno používat automatizované DD: prostředky databází nebo v lepším případě integrovanou součást CASE prostředků, nebo při nejhorším alespoň textový procesor.

4.3.4 Závěr ke slovníku dat

Tvorba DD je jednou z časově nejnáročnějších činností při analýze a návrhu systému. Ale zároveň jednou z nejdůležitějších činností. Bez formálního slovníku, který definuje význam všech pojmů, nemůže být popis systému precizní a pravdivý.

4.4. Funkční model - specifikace procesu (minispecifikace)

Minispecifikace je popis procesu na nejnižší úrovni hierarchického rozkladu. Upřesňuje, jaká je logika procesu (co se dělá když ..., jak dlouho apod.). Tedy určuje, co se musí udělat při transformaci vstupů na výstupy. Jde o detailní popis činnosti uživatelské organizace, kterou popisují jednotlivé elementární DFD bubliny. Pozor! Nejde o strukturu programu!

Úrovně DFD nepopisují o systému vše. Vyjadřují rozdělení systému na části tak, aby každá část mohla být specifikována konzistentně a zároveň nezávisle na ostatních částech systému, a zobrazují rozhraní mezi částmi. Nejmenší části systému jsou elementární funkce (na nejnižší úrovni DFD). Tyto funkce (a pouze tyto) musí být specifikovány tak, aby dohromady popisovaly celý systém. Funkce vyšších úrovní nemá význam specifikovat, protože nejsou nic jiného, než množina funkcí nižší úrovně.

Pokud dosáhneme úrovně elementárních funkcí, není těžké každou takovou funkci popsat přirozeným jazykem. Ale přirozený jazyk je nebezpečný, protože může vést k nejednoznačnosti výkladu, může být nekonzistentní, zavádějící a neúplný. Proto by prostředky pro specifikaci procesu měly splňovat tyto požadavky:

- musí existovat **jedna minispecifikace pro každou elementární funkci** z množiny DFD,

- minispecifikace **musí vyjadřovat postup**, jak jsou datové toky, které do funkce vstupují, **transformovány** na výstupní datové toky funkce,
- minispecifikace musí vyjadřovat, co transformace znamená. Nemusí vyjadřovat způsob pro implementaci transformace,
- minispecifikace by se měla snažit o kontrolu redundance. Tj. neměla by znovu vyjadřovat to, co je již v DFD nebo v data dictionary (i když malá redundance je možná, jestliže zlepší čitelnost minispecifikace),
- množina výrazů pro vytvoření minispecifikace by měla být jednoduchá a ne příliš rozsáhlá. Navíc by nás měla nutit, abychom se vyjadřovali standardním způsobem, - minispecifikace musí být srozumitelná jak uživateli, tak i analytikovi.

Minispecifikace lze vyjádřit různě:

- strukturovanou angličtinou (češtinou), tj. pseudokódem,
- rozhodovací tabulkou,
- rozhodovacím stromem,
- i strukturním diagramem, Nassi-Schneidermanovým diagramem ...

Minispecifikace musí být v takové formě, aby mohla být ověřena uživatelem (aby tomu rozuměl uživatel, vedoucí, kontrolor, ten, kdo testuje). Musí umožnit vyjádřit sekvenci, selekci a iteraci, logické operace.

Strukturovaný přirozený jazyk (angličtina, čeština, japonština)

Tento upravený jazyk jako prostředek specifikace je omezenou verzí přirozeného jazyka. Do něj jsou začleněny jednoduché konstrukce ze strukturovaného programování. Slovník tohoto jazyka by měl zahrnovat:

- rozkazovací tvary sloves (přídavná jména a příslovce vedou k subjektivnímu výkladu a mohou zavádět),
- termíny z data dictionary,
- rezervovaná slova k vyjádření logiky.

Syntax strukturovaného přirozeného jazyka předepisuje:

- jednoduché (holé) věty,
- uzavřené rozhodování (jednoznačné vyjádření selekce),
- uzavřené opakování (jednoznačné vyjádření cyklu),
- kombinace předchozích tří konstrukcí.

Strukturovaný přirozený jazyk je třeba odlišovat od pseudokódu na úrovni programovacího jazyka. Tento pseudokód je jazykový prostředek, který se používá zvlášť v modulárním programování, ale též pro specifikaci modulu a pro údržbu modulu. Je to neformální a velmi flexibilní programovací

jazyk, který není závislý na konkrétním počítači, ale používá se k vyjádření myšlenek programátora před kódováním. Existuje několik notací pseudokódu. Je založen na přirozené angličtině, i když využívá konstruktů obvyklých v programovacích jazycích 3.generace. Je méně srozumitelný uživateli (hlavně běžnému českému). Vznikl v oblasti strukturovaného programování a používá se i ve structured designu k vyjádření vnitřní činnosti modulů.

Jestliže se analytik rozhodne používat strukturovaný přirozený jazyk, musí vytvořit nebo převzít slovník a syntax takového jazyka. Pro představu uvedeme příklad strukturované angličtiny a češtiny. Strukturovaná angličtina v pojetí z našeho příkladu je již blíže k "programovacímu" pseudokódu než k přirozenému jazyku.

Příklad 1: Strukturovaná angličtina (s prvky pseudokódu).

Sekvence - dána pořadím zápisu operací.

Selekce - `IF podmínka`

```
      THEN ... (operace)
   ELSE ...
   ENDIF

   DO CASE
      CASE ...
   ENDCASE
```

Iterace - `DO WHILE podmínka`

```
      ...
   ENDDO
```

Aritmetické a logické operace:

Např.: `X = A+B/C*(D+X)`

Další rezervovaná slova:

```
GET (nebo ACCEPT nebo READ)
PUT (nebo DISPLAY nebo WRITE)
FIND (nebo SEARCH nebo LOCATE)
DELETE
VALIDATE
REPLACE
MOVE
SET
SORT
```

Příklad:

```
    GET objednavka record
IF (cislo-objednavky in objednavka record =
     cislo-objednavky in objednavka)
  THEN ...
       ...
  WRITE ok-objednavka
ENDIF.
```

Příklad 2: strukturovaná čeština

Sekvence - za sebou psané holé věty.

Iterace - Pro každý ... proveď:

Selekce - Jestliže ...
potom ...
jinak ...

Další rezervovaná slova
zapiš, čti, nastav ...

Příklad:

2.3.2. VYSTAVENÍ FAKTURY ZÁKAZNÍKOVI
Pro každou objednávku zákazníka proveď:
1. Zapiš jméno a adresu zákazníka do faktury.
2. Jestliže zákazník patří do kategorie "SPECIALNÍ"
potom 2.1. Čti slevu ze storu SLEVY podle speciálního indikátoru
jinak 2.2. (běžný zákazník) nastav slevu na 0.
3. Pro každou položku prodeje na objednávce zákazníka proveď:
3.1. Zkopíruj na fakturu číslo zboží a objednané množství.
3.2. Čti cenu za jednotku ze storu CENY podle čísla zboží.
3.3. Nastav položku mezisoučet na (cena za jednotku x objednané množství x (100 - sleva) %).
4. Nastav součet za fakturu na (suma položek mezisoučtů).
5. Nastav celkový dluh na (součet za fakturu - splacený dluh)

Výhodou českého popisu je lepší srozumitelnost uživateli, ale pro češtinu zatím neexistuje žádný generátor.

Neexistují přesná pravidla pro strukturovaný jazyk. Analytik musí volit mezi pevným a přesným popisem na jedné straně a srozumitelností uživateli na straně druhé. Nesmí trvat na dané formě, zvlášť když nechce uživatele zastrašit strohostí.

V žádném případě však nepoužívat skoků (skoky zavádějí pozornost).

Rozhodovací stromy (decision trees)

Strukturovaný jazyk se pro některé typy transformací nehodí. Představme si situaci, že akce uživatele závisí na několika možnostech, které mohou dohromady tvořit velký počet kombinací. Slovní popis by byl zavádějící a asi hodně hluboko vnořovaný. Zde je vhodný rozhodovací strom.

Rozhodovací strom jako grafický prostředek odděluje nezávislé podmínky a ukazuje činnosti, které se provádějí na základě každé možné kombinace.

Rozhodovací strom se čte zleva doprava. Výsledek každého rozhodnutí vede vždy na jednu větev stromu.

Rozhodovací tabulka

Rozhodovací tabulka je tabulkovou formou rozhodovacího stromu. Lze ji lépe uchovat v textové, počítačem čitelné podobě. Ale na druhé straně je na první pohled trochu méně srozumitelná uživateli.

Jiné formy specifikace procesu

Specifikaci procesu lze vyjádřit např. i obrázkem (např. strukturou vytvářeného výstupu). Spolu s DD bude naprosto jasná, víc než jakýkoliv slovní popis. Lze použít i formuláře, výtahy z existujících uživatelských příruček, grafy apod. Možnosti jsou neomezené. Často však závisí i na tom, jak chceme specifikaci dále použít. Pokud používáme CASE systém, může mít určité prostředky pro specifikaci zabudované. Pak je vhodné tyto prostředky využít, zvláště pokud mají vazbu na další kroky vývoje systému (např. generování zdrojových kódů programů).

Závěr k minispecifikaci.

Existuje řada různých způsobů, jak popsat podrobné činnosti uživatele pro každou "elementární bublinu v DFD". Strukturovaná angličtina se v anglicky mluvících zemích používá nejčastěji, ale lze použít libovolný z výše uvedených prostředků i prostředky další.

Specifikace procesů zahrnuje největší objem podrobné práce při vytváření modelu systému. Opět v jednom systému mohou být stovky i tisíce minispecifikací. Proto implementace může a vlastně i musí začít dřív, než jsou všechny minispecifikace hotové.

Minispecifikace je také "testem" DFD - mohou být odhaleny další vstupní nebo výstupní datové toky, které v DFD chybí nebo další funkce, které jsou potřeba. (Například při tvorbě minispecifikace funkce, která přidává nová data do storu ZAKAZNICI, zjistíme, že chybí funkce, která tato data mění nebo ruší).

4.5. Diagramy přechodu stavů - STD

STD (State-Transition Diagrams, diagramy přechodu stavů) slouží k modelování chování systému, které závisí na čase. Neobejdeme se bez nich při návrhu real-time systémů. Důležitou součástí specifikace real-time systémů je totiž popis, co se kdy stane. Tento popis lze vyjádřit STD.

Notace STD

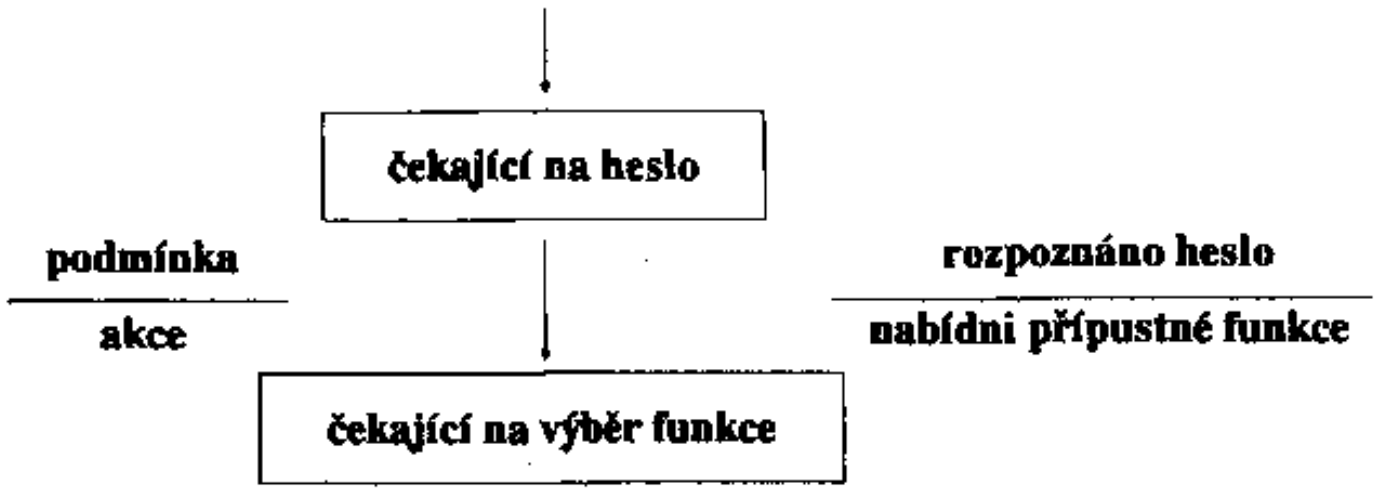
Hlavními komponentami diagramu jsou obdélníky reprezentující stavy a šipky vyznačující změny stavů.

Stav systému

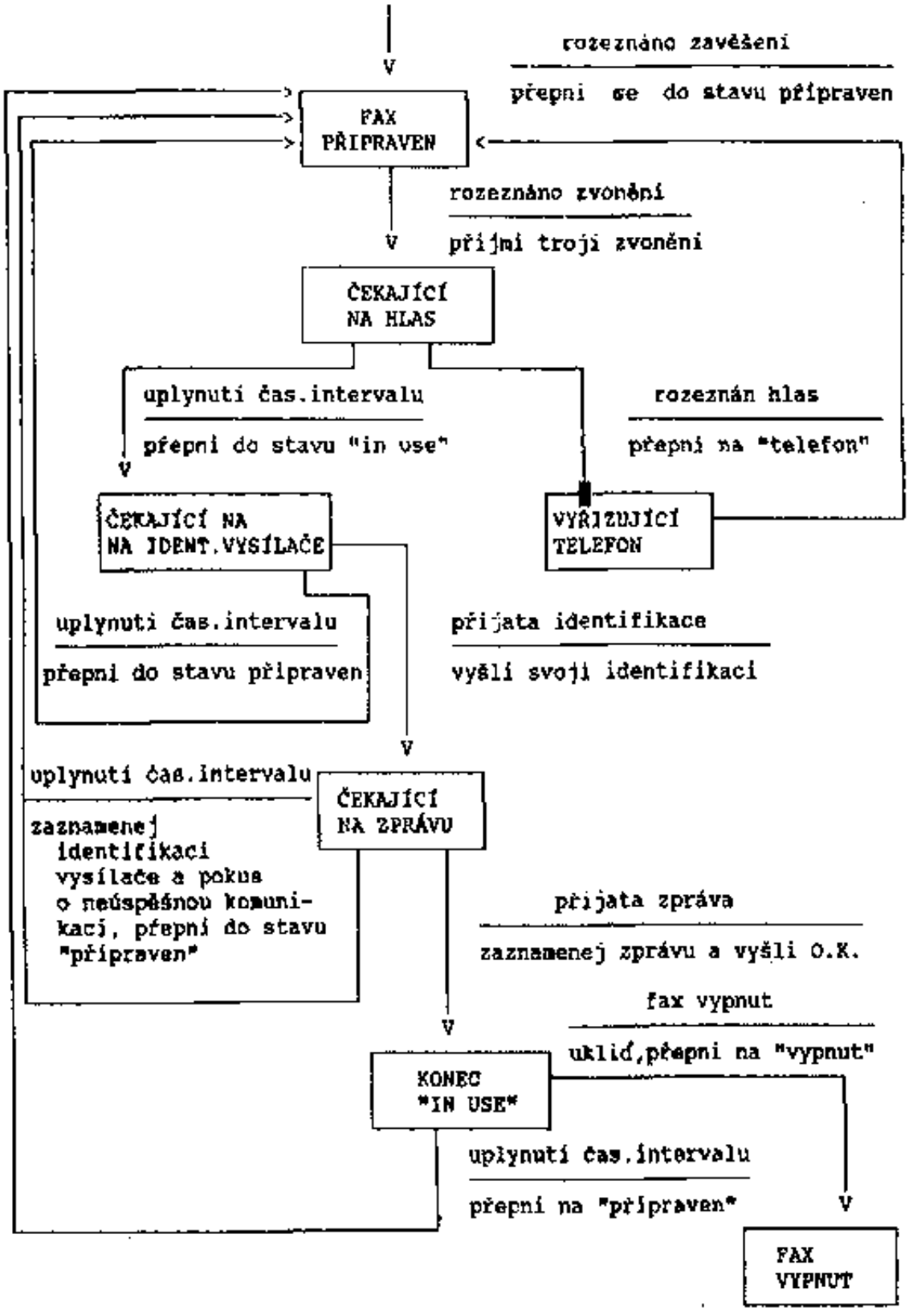
Stav systému pojmenovává akční vlastnosti systému v daném okamžiku: např. "systém v čekání na zadání hesla", "systém připraven k přijetí příkazu" apod..

Změny stavu

Změna stavu znamená přechod modelovaného systému z jednoho důležitého rozpoznatelného stavu do dalšího. Změna stavu nastane při rozpoznání, že je splněna určitá podmínka. Příklady podmínek: "přijata zpráva", "uplynul stanovený čas" apod. Ze stavu do stavu přejde systém provedením patřičných akcí: "zaznamenej identifikaci vysílače a přepni se do stavu připraven", "uklid' a přepni se do stavu vypnut" apod. Podmínky a akce se zachycují v STD jako popis šipek (změn stavů):



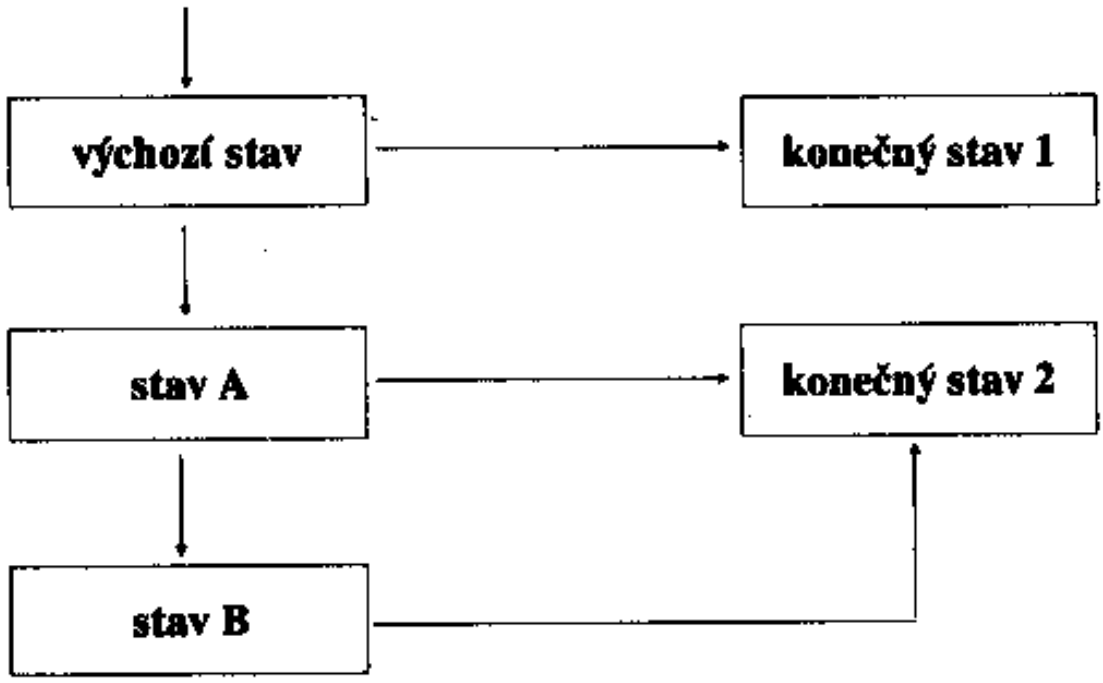
Příklad ilustrující celou notaci STD je na obr. 12.



Obr. 12. STD - Příklad diagramu pro fax (příjem signálu, zjednodušeno)

Výchozí a konečný stav v STD

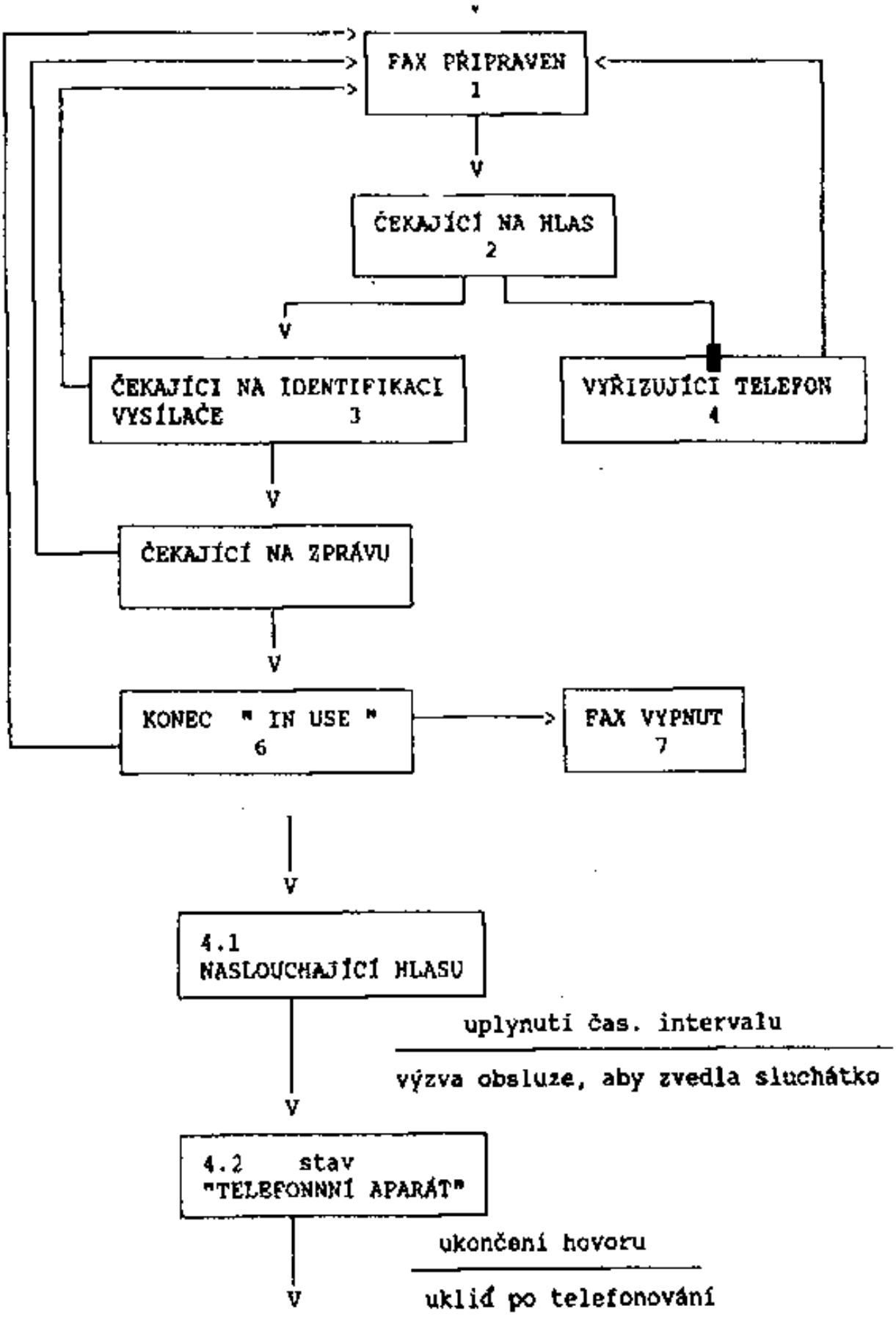
Modelovaný systém musí mít definován výchozí stav (initial state). Proto má STD jeden "kořen" - jeden vstupní obdélník. Konečných stavů systému (final states) může být více. V notaci jsou výchozí a konečné stavy rozlišitelné podle šipek - ilustrace viz obr. 13. Výchozímu stavu žádný stav nepředchází, po konečném stavu žádný další nenásleduje.



Obr. 13. STD - Notace výchozího stavu a konečných stavů

Dělení rozsáhlých STD, hierarchizace STD

Příklad na obr. 12 je velmi zjednodušen, a přesto se část "vyřizující telefon" již rozumně na stránkový formát obrázku nevešla. Reálný systém má obvykle desítky stavů, jež je třeba rozlišit. V tomto případě používáme členění a hierarchizaci diagramů obdobnou jako u DFD. Výrazná šipka u komponenty "vyřizující telefon" v obr. 12 naznačuje, že tato komponenta bude rozložena jako samostatný STD nižší úrovně. Vazbu mezi úrovněmi je vhodné zviditelnit číslováním stavů. Zásady číslování jsou analogické jako u DFD. Hierarchický rozklad i číslování si můžete prohlédnout na obr. 14.

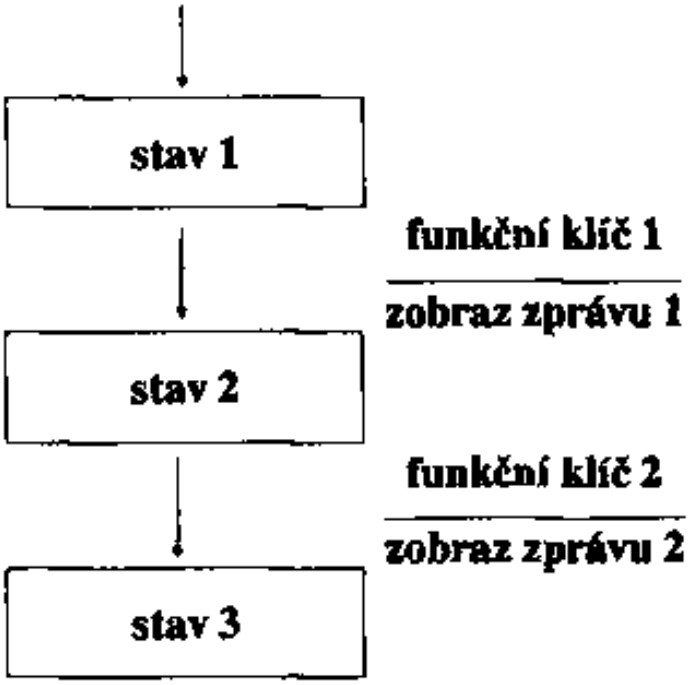


Obr. 14. STD - Hierarchizace STD a číslování stavů

Kontrola konzistence STD

YSM neposkytuje návod, jak odvodit stavy modelovaného systému. Jestliže jsme však již STD odvodili, můžeme zkontrolovat jejich konzistenci dle jednoduchých, převážně formálních pravidel:

- jsou definovány všechny stavy? Simulujte si na STD průchod všemi větvemi a konfrontujte tuto simulaci s chováním reálného systému. Možná objevíte chybějící stav či nesprávnou podmínku, akci;
- nechybí v STD změna stavu (šipka)? Lze se dostat do každého ze stavů?
- reaguje v daném stavu systém na všechny přípustné podmínky? Reaguje adekvátními akcemi? V lit. (YSM) se uvádí v tomto kontextu příklad z obr. 15.

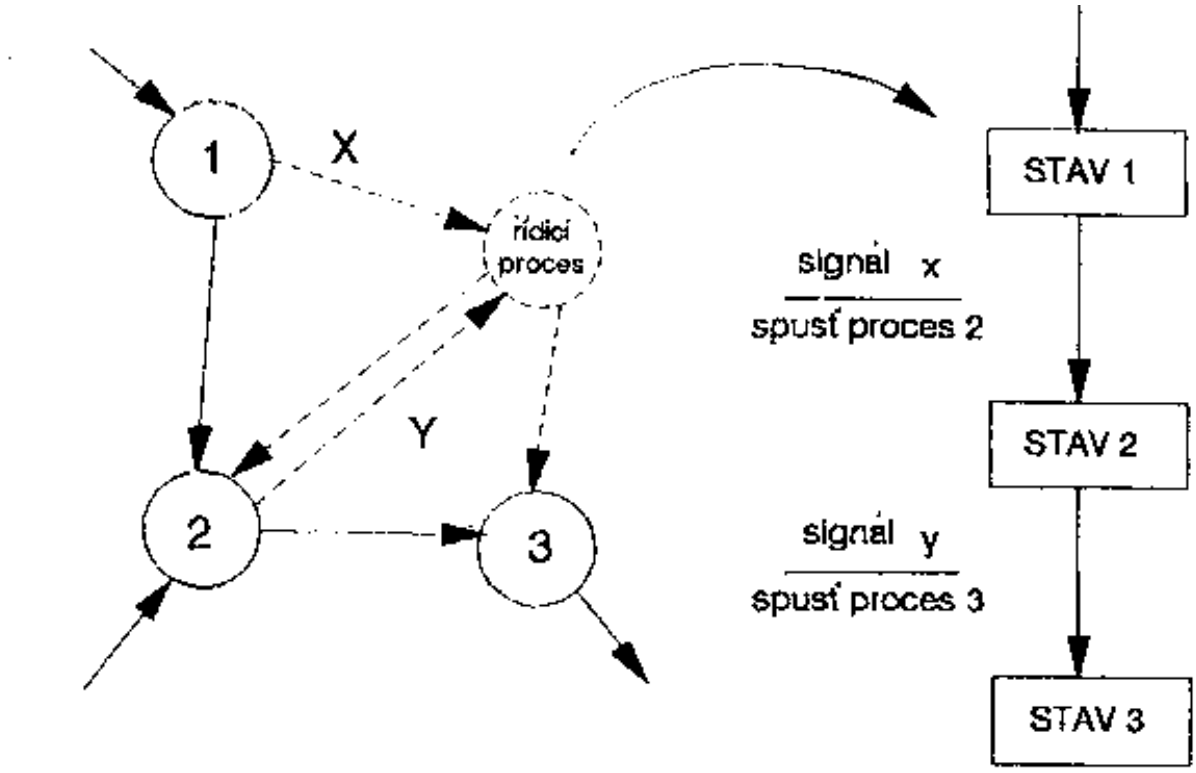


Obr. 15. STD - Neúplný diagram

Systém zhotovený podle obr. 15 bude fungovat správně jen tehdy, jestliže ukázněný uživatel bude mačkat právě jen pf-klíč 1 a pf-klíč 2, a to jen v patřičný okamžik. Na jakoukoli jinou situaci není takto navržený systém schopen reagovat.

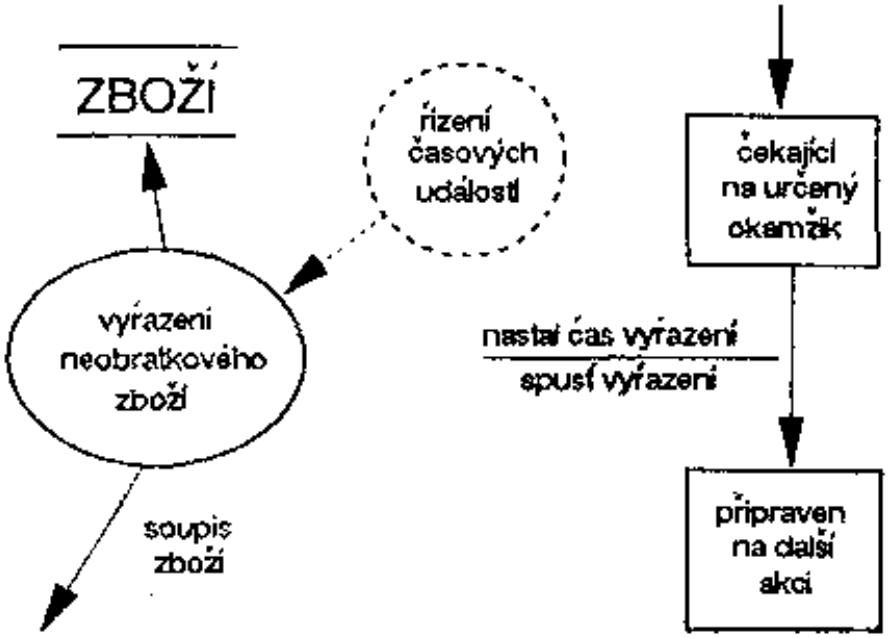
Vztah STD a DFD

Při nejběžnějším použití reprezentuje STD řídící proces z DFD. Schematicky je tato situace vidět na obr. 16.



Obr. 16. STD - Vztah mezi STD a DFD

Konkrétní příklad je na obr. 17.



Obr. 17. STD - Vztah DFD a STD na konkrétním příkladu

Kdy použít STD jako nástroj modelování

Pro systémy reálného času je vhodné STD použít jako základní nástroj. Ve složitých ekonomických aplikacích je často základním problémem zajištění doby odezvy. Při modelování těchto "real-time" prvků rovněž velmi pomáhá STD.

4.6. Vztahy mezi nástroji

Každý uvedený nástroj modelování IS je zaměřen na jiný aspekt modelovaného systému (DFD na funkce - procesy a toky dat mezi procesy, ERD na datové objekty a vztahy mezi nimi, STD na koordinaci a řízení procesů probíhajících v reálném čase, tedy na časové charakteristiky systému).

Jak spolu uvedené nástroje souvisí? A souvisí spolu vůbec, nebo jsou nezávislé? Ve velkých projektech je vážné nebezpečí, že právě mezi modely vznikne řada nekonzistencí, duplicit, synonym apod. Stejné nebezpečí hrozí, pokud se na projektu účastní lidé s různým zázemím (nejednotnost pojmů). Tím vzniká i nebezpečí chyb a zvýšení nákladů při údržbě.

Proto je třeba ověřit konzistenci každého modelu a mezi modely navzájem. Takto ověřený systém je možné označit za vyvážený systém.

Dobrý CASE systém by měl takovou konzistenci kontrolovat a pomáhat ji udržovat!

a) DFD - a DD

- Každý datový tok a každý data store musí být definovány v DD. Pokud jejich definice chybí, je datový tok nebo data store považován za nedefinovaný.
- Naopak - každý datový prvek a každý data store definovaný v DD se musí objevit někde v DFD. Jinak se jedná o "ducha" = něco, co je definované, ale není použité.

b) DFD - a minispecifikace

- Každá "bublina" v DFD musí mít podřízený DFD diagram nebo minispecifikaci, ale ne obojí - potom by byl model zbytečně (a také nebezpečně) redundantní (**Pozor**, že např. CASE SDW umožňuje obojí).
- Každá minispecifikace musí mít elementární proces v DFD ("trampující proces").
- Musejí souhlasit vstupy a výstupy. Pro každý datový tok vstupující do elementární funkce v DFD musí v odpovídající minispecifikaci existo-

vat operace READ (nebo čti nebo get nebo accept apod.), pro každý výstupní datový tok operace WRITE (nebo zapiš nebo put apod.).

c) ***Minispecifikace - a DFD a DD***

- Každý odkaz na data (podstatné jméno) v minispecifikaci je odkaz buď na datový tok nebo store, na který se elementární funkce odvolává, nebo
- je lokálním pojmem, explicitně definovaným v minispecifikaci, nebo
- je součástí datového toku nebo storu spojeného s elementární funkcí (musí být vidět v DD). Např. v minispecifikaci se používá X a Y, vstupem do funkce je ale Z. Pokud je v DD např. Z = X + Y, je to v pořádku.

d) ***DD a - DFD a minispecifikace***

DD je konzistentní, pokud platí následující pravidlo:

- Vše, co je definováno v DD, musí být použito v minispecifi- kaci, nebo v DFD, nebo v jiné definici DD.

Pokud modelujeme i stávající implementaci systému, mohou se v DD vyskytovat i data, která se nebudou dále používat.

e) ***ERD a - DFD a minispecifikace***

- Každý data store v DFD musí korespondovat s entitou (typem), nebo se vztahem, nebo s kombinací entity a vztahu z ERD. Pokud je v DFD store, který nesouvisí s datovým modelem, a nebo pokud je v ERD entita, která není součástí žádného storu, něco není dobře.
- Názvy entit a storů musejí jít dohromady (ZAKAZNICI = {ZAKAZNIK})
- Popisy v DD se musí odvolávat jak na DFD, tak na ERD. Např. ZAKAZNICI = {ZAKAZNIK} ZAKAZNIK = jmeno + adresa + telefon + ...

Yourdon doporučuje používat pro typ entity jednotné číslo, pro data store množné číslo (je to souhrn výskytů).

- V minispecifikacích musí být pro každou entitu a vztah vyskytující se v ERD operace vytváření a rušení.
- Každý atribut má v alespoň jedné elementární funkci nastavenou hodnotu a alespoň jedna elementární funkce atribut (datový prvek) používá (čte).

f) DFD - a STD (state-transition diagram)

- Každý řídící proces v DFD má svůj STD , který tento proces specifikuje. A naopak, každý STD patří k jednomu řídícímu procesu z DFD.
- Každá podmínka v STD musí korespondovat s jedním vstupním kontrolním tokem do daného řídícího procesu. A naopak, každý vstupní řídící tok do řídícího procesu musí mít podmínku v korespondujícím STD.
- Každá akce v STD musí korespondovat s výstupním řídícím tokem z řídícího procesu, k němuž STD patří. A naopak, každý výstupní řídící tok z řídícího procesu musí souviset s příslušnou akcí korespondujícího STD.

Poznámka na závěr: Provést poctivě ověření všech modelů navzájem je ručně velice pracné. Průměrné prostředky typu CASE by měly mít tyto kontroly zabudované a měly by je provádět automaticky (jak při vytváření návrhu, tak, a to hlavně, při jeho opravách). Je však potřeba znát základní principy automatických kontrol (Proč?). A k tomu sloužil tento výklad.