A P P L I C A T I O N M A S T E R

A. Adámek

1. Úvod

Rychlý rozvoj výpočetní techniky způsobil, že výpočetní střediska na celém světě i u nás stojí před problémem rostoucích požadavků uživatelů na zpracování dat. Problémy se vyskytují z těchto důvodů:

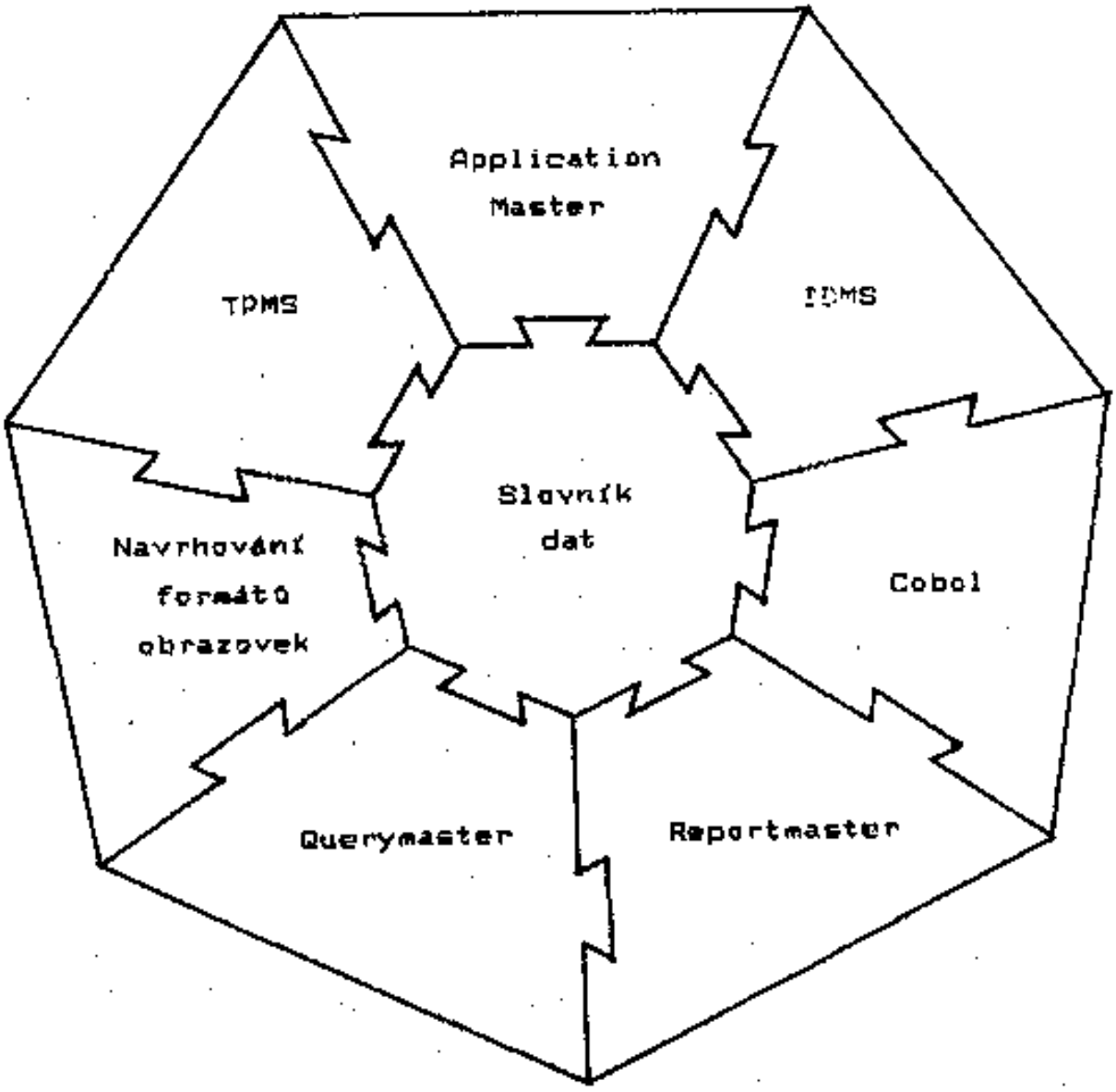
1. Počet a schopnosti počítačů rostou rychleji než počet a schopnosti programátorů
2. Roste složitost vyvíjených aplikací
3. U zavedených středisek více než polovina programátorských zdrojů se spotřebuje na modifikace a opravy existujících aplikací.
4. Existuje nedostatek kvalitních odborných pracovníků

Mnoho systémů zpracování dat nesplňuje potřeby koncového uživatele. (Koncovým uživatelem nazýváme bezprostředního uživatele systému zpracování dat, jako je účetní, plánovač, vedoucí). Nejdůležitějším důvodem, proč je koncový uživatel nespokojen bývá nesoulad mezi systémem zpracování dat a reálným prostředím, které má popisovat. Organizace, metody práce i požadavky uživatele jsou předmětem neustálých změn. Chce-li být systém zpracování dat pro organizaci opravdu přínosem, musí se také měnit a to stejnou rychlostí jako organizace, a ne s několikaměsíčním zpožděním.

Tradiční způsob vývoje informačních systémů je založen na používání jazyků vyšší úrovně jako je např. Cobol, a na nasazení moderních technik jako je metoda strukturovaného návrhu. Tyto metody přinesly během posledních desetiletí zvýšení produktivity, bohužel jen částečně. Ve snaze získat svůj systém zpracování dat co nejrychleji, snaží se řada organizací o získání už existujících systémů. Některé organizace modifikují takto získané systémy k obrazu svému, za předpokladu, že s produktem získají i zdrojový text a s vědomím, že od této chvíle jsou odpovědny za údržbu produktu. Nicméně mnoho instalací musí vyvíjet nové systémy samo.

Mnoho firem ve světě se snaží vyvíjet produkty a jazyky, které by uživatelům vývoj nových aplikací usnadnily. Pozadu nezůstává ani firma ICL, která dodává ke svým počítačům tzv. prostředí vývoje aplikací (ICL application development environment), jehož zajímavou součástí je produkt Application Master (AM).

2. Prostředí vývoje aplikací firmy ICL



Prostředí vývoje aplikací firmy ICL je reprezentováno řadou produktů, které mají usnadnit a urychlit vývoj nových aplikací, jejich údržbu a umožnit snadný přístup k informacím uložených v databázi IDMS. Firma ICL se pokouší zákazníky přesvědčit, že

snížení nároků na vývoj, implementaci a následnou údržbu aplikací je desetinásobné ve srovnání s běžným Cobolovským přístupem.

Ústředním prvkem tohoto prostředí je slovník dat (Data Dictionary System). Tento slovník dat je vlastně databáze používaná pracovníky při vývoji aplikace, tj. analytiky a programátory. Jsou v ní informace o všech formátech vět a datových položek, formátech obrazovek, o struktuře databáze, o formátech sestav atd. Firma dodává několik procedur, které umožňují práci se slovníkem dat, jako aktualizaci, výpisy. Obsahem slovníku dat jsou věty, kterým se říká elementy. Existují elementy různých typů. Na příklad, element RECORD PRAC-VETA popisuje větu PRAC-VETA , nebo element ITEM POR-CIS popisuje datovou položku POR-CIS. Je možno použít asi 30 typů elementů, další jsou např. SCHEMA, SUBSCHEMA, GROUP, TASK, MODULE, FILE. Každý element se popíše pomocí svých parametrů. Pro každý element existuje seznam parametrů, s předdefinovanými nejpoužívanějšími hodnotami. Takovými parametry pro typ elementu RECORD jsou na příklad LENGTH, STRUCTURE, KEY. Je pamatováno i na stadium analýzy. Analytik disponuje několika typy elementů, kterými může popsat data a procesy v projektu. Tyto elementy pak mohou být svázány s elementy popisujícími odpovídající reálná data, se kterými pracuje programátor.

Všechny ostatní prvky vývojového prostředí se slovníkem dat úzce spolupracují. Programy v Cobolu si z něho berou popisy vět. Databáze IDMS se generuje přímo z popisu ve slovníku, podobně jako TPMS (Transaction Processing Management System), což je systém řízení interaktivních terminálově založených aplikací. Querymaster a Repotmaster umožňují rychle získat požadované výstupy z databáze buď na obrazovku nebo na tiskárnu. Všechny tyto produkty mají výhodu vyplývající z faktu, že veškerá data o aplikaci jsou již zavedena ve slovníku dat, takže stačí např. uvést, že chceme zobrazit údaje CIS-PRAC, JMENO-PRAC a HRUBA-MZDA a Querymaster ví kde má tyto údaje v databázi hledat.

Dalším důležitým prvkem vývojového prostředí ICL je interaktivní navrhování formátů obrazovek. Formát obrazovky je jedním z elementů slovníku dat. Při jeho navrhování se napíše formát přímo na obrazovku v požadovaném tvaru včetně hlaviček a popisných

informací, a dále se vyznačí začátky jednotlivých vstupních, výstupních i modifikovatelných údajů. Tyto údaje pak systém probírá a dotazuje se na jejich vlastnosti. Obvykle stačí udat jejich název ze slovníku dat a systém si ostatní údaje sám doplní. V tomto stadiu je možné určit způsob zobrazení údaje, způsob jeho validace a způsob oznámení chyby.

3. Application master (AM)

Takovýto slovník dat není nic převratného, firma ICL jej má již léta, a podobné prostředky existují i u jiných firem. Nyní jej však rozšířili o specifikaci procesů, takže vznikl vlastně slovník dat a procesů. Z procesů popsaných ve slovníku dat pak Application Master produkuje výsledný program. Tyto procesy zahrnují jednak interakci s koncovým uživatelem, který typuje data na terminálech, jednak činnosti pojednávající s daty v databázi. Specifikace procesů je popsána jazykem velmi vysoké úrovně, podobně jako u jiných elementů ve slovníku dat, formou parametrů. Posledním pozůstatkem procedurálního programování je parametr PROCESSING-STRUCTURE. Procesy jsou popsány pomocí dvou typů elementů (ve skutečnosti jsou tři) ve slovníku dat. Tyto elementy jsou pojmenovány DIALOGUE a EXCHANGE.

Je-li terminálově založená aplikace v chodu, vidíme, že se skládá ze série interakcí mezi koncovým uživatelem (operátorem terminálu) a aplikací. Zobrazí se obrazovka, na které jsou popisné informace a data z databáze a dále vstupní pole, do kterých uživatel typuje své data. Po natypování dat uživatel stlačí tlačítko SEND, tím pošle data aplikaci a v ní budou data zpracována. Této sekvenci - zobrazení dat, natypování dat a zpracování dat se zde říká EXCHANGE (výměna). Série takových výměn se nazývá DIALOGUE (dialog).

3.1 Element DIALOGUE

Tento element v podstatě pouze popisuje strukturu podřízených procesů DIALOGUE a EXCHANGE. Tyto procesy mohou být v následujících strukturách:

- posloupnost

- volba jednoho z více procesů
- opakování jednoho procesu

V tomto elementu může existovat i parametr TASK, který pojemnová- vá modul vytvořený kompilací AM. Parametr FAIL umožňuje pro každý proces specifikovat, kdy má být proces považován za chybný a jaká akce se má v tomto případě provést. Parametr SUBSCHEMA informuje AM o názvu subschéma IDMS, kde jsou popsány struktury vět a jejich vazby. Toto subschéma platí i pro podřízené procesy.

3.2 Element EXCHANGE

Tento element popisuje proces odehrávající se při zpracování jedné obrazovky pomocí následujících parametrů:

1. DISPLAY-STRUCTURE.

Určuje název formátu obrazovky, který je uložen ve slovníku dat.

2. PROCESSING-STRUCTURE.

Zde se definuje pořadí činností:

- výstup-vstup z terminálu
- čtení z databáze
- psaní do databáze
- volání podprogramu psaného např. v Cobolu
- činnosti nad lokalními proměnnými

Obvykle se tento parametr vypouští, předdefinováno je pořadí prvých tří uvedených činností. Pokud nejsme schopni zajistit požadované funkce skrze prostředky AM, lze napsat vlastní podprogramy, které voláme na kterémkoliv místě v PROCESSING-STRUCTURE. Data do a z podprogramu se předávají pomocí parametrů.

3. SINGULAR-VIEW

Zde se uvede seznam vět, které budou čteny z databáze, případně vyhledávací algoritmus.

4. SELECTION.

Uvádí podmínku, podle které se vybírá čtená věta

5. CHANGES.

Zde se definují akce prováděné při zápisu do databáze. Může se určit tvorba nové věty, modifikace věty nebo výmaz.

3.3 Kompilace AM

Jsou-li všechny procesy zapsány ve slovníku dat, je možno kompilovat. Kompilátoru zadáme název dialogu na nejvyšší úrovni a výsledkem kompilace je jeden výsledný modul s názvem, který je uveden v parametru TASK dialogu na nejvyšší úrovni. Tento modul pak obsahuje veškeré podřízené dialogy a výměny. Chceme-li výsledek rozdělit do více modulů, uvedeme parametr TASK i do těch dialogů, pro které chceme vytvořit samostatný modul. Pak je potřeba provést tolik kompilací, kolik dialogů s TASKem v aplikaci existuje.

Veškeré elementy ve slovníku dat, tudíž i procesy DIALOGUE s EXCHANGE mohou obsahovat číslo verze dané parametrem VERSION. Tímto způsobem lze mít ve slovníku dat více verzí zároveň.

4. Příklad jednoduché aplikace

V této kapitole je uveden příklad velmi jednoduché aplikace. Cílem je ukázat možnosti AM aniž by bylo podáno detailní vysvětlení. Tento jednoduchý příklad se týká vyhledávání a aktualizace zákazníka. Informace o zákazníkovi obsahuje číslo zákazníka, jméno, adresu a úvěrový limit. Tato aplikace má umožnit vyhledání zákazníka podle čísla a změnu kteréhokoliv údaje, s výjimkou čísla zákazníka. Pokud již nechceme měnit údaje u žádného dalšího zákazníka, typujeme "DOST" namísto čísla zákazníka.

4.1 Datová struktura

Předpokládejme, že v našem datovém modelu existuje v databázi o zákaznících jediný typ věty. Každá taková věta obsahuje několik údajů, např.

- číslo zákazníka
- jméno zákazníka
- adresa - 1. část
- adresa - 2. část
- směrovací číslo
- úvěrový limit

Struktura věty zákazníka musí být popsána ve slovníku dat a musí být vytvořená databáze s větami zákazníků. Ve slovníku dat je subschéma popsáno pod názvem AMSUBSCH, věta zákazníka je popsána s názvem VETA-ZAKAZNIKA a její struktura je určena údaji, které jsou ve slovníku dat poptásy jako CISLO-ZAKAZNIKA, PRIJMENI-JMENO, ADRESA-1, ADRESA-2, SMER-CISLO a UVRR-LIMIT. Každý z těchto údajů je blíže specifikován svými parametry, na příklad CISLO-ZAKAZNIKA má parametr PICTURE X(5).

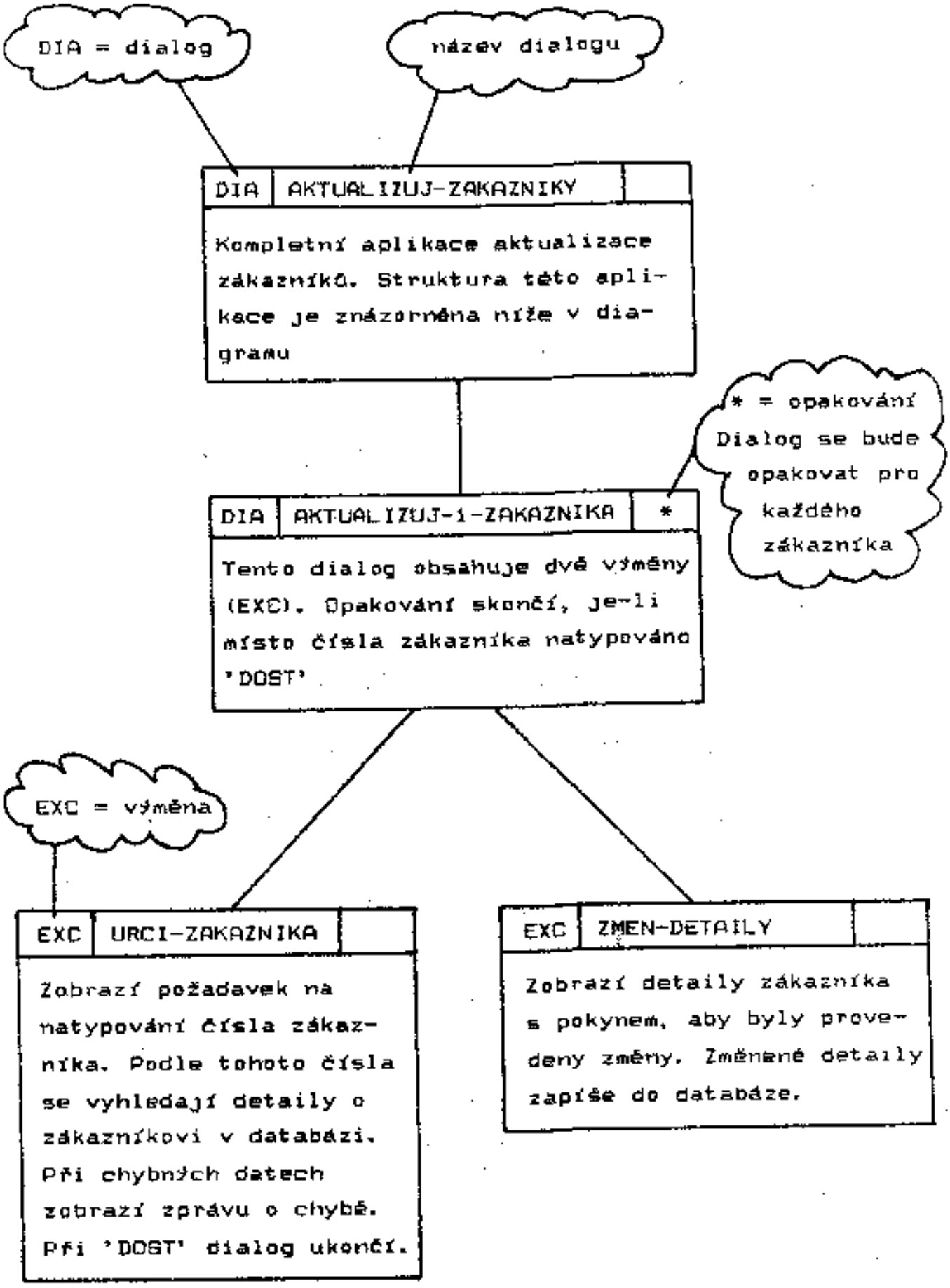
4.2 Strukturovaný návrh aplikace

Aplikaci lze tedy popsat jako strukturu dialogů a výměn a lze ji znázornit i graficky. Vznikne strukturovaný diagram, ve kterém kompletní aplikace může být znázorněna jako dialog na nejvyšší úrovni. Struktura tohoto dialogu je rozepsána na druhé úrovni diagramu. Podobně lze rozepsat strukturu dialogů druhé úrovně pomocí třetí úrovně, atd., dokud neznázorníme kompletní strukturu aplikace. Strukturovaný diagram našeho příkladu má tři úrovně, jak možno vidět na následujícím grafu.

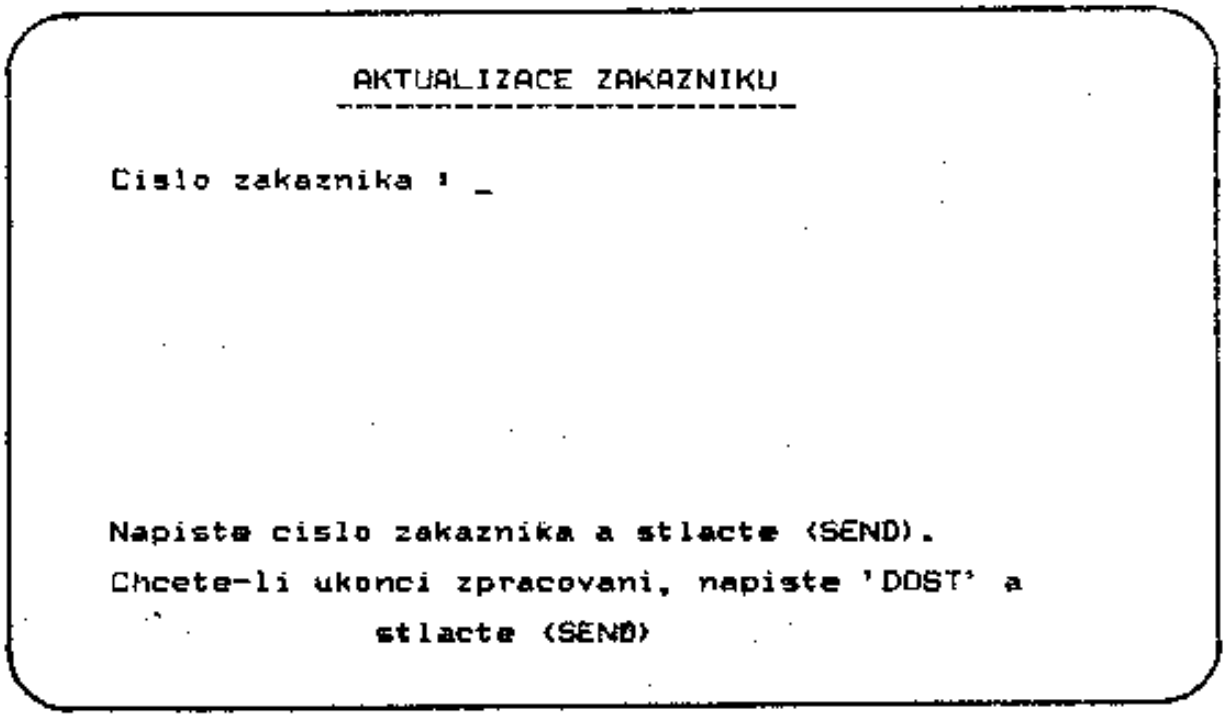
4.3 Návrh formátu obrazovky

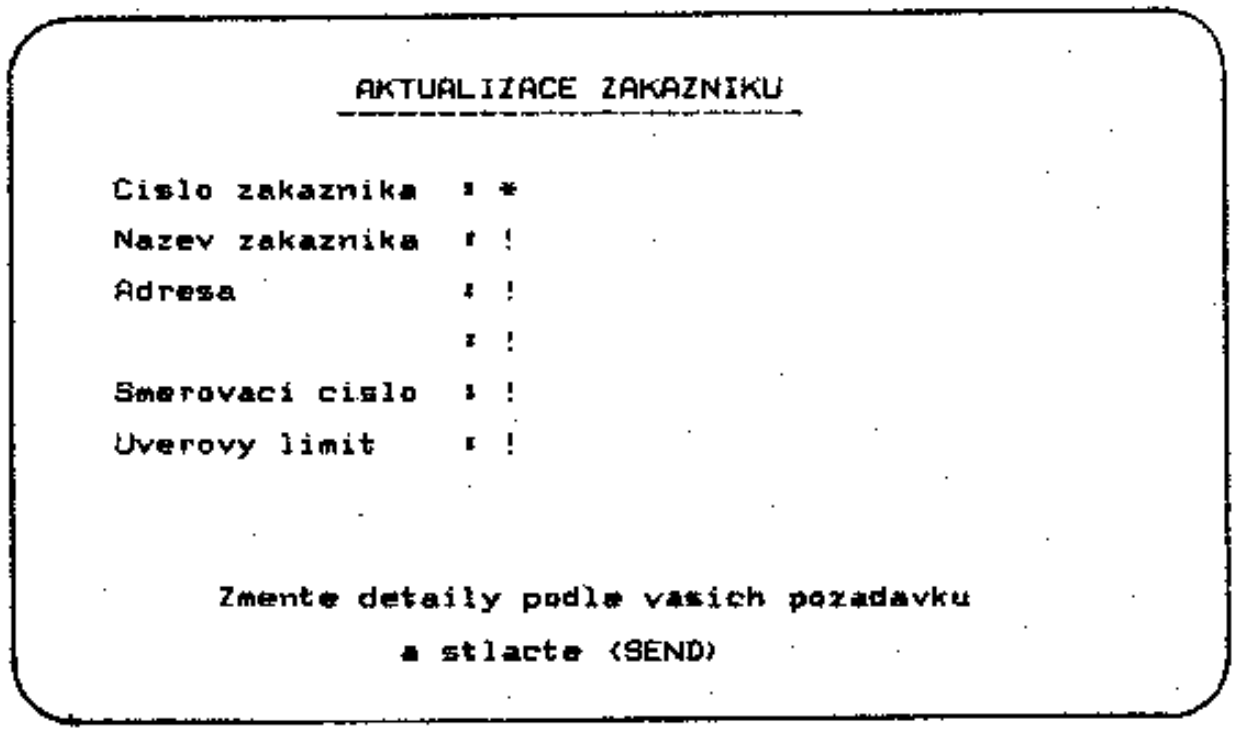
Je-li struktura aplikace navržena, pokračujeme návrhem formátů obrazovek, obvykle jeden formát na každou výměnu (EXCHANGE) ve strukturovaném diagramu. Pojmenované formáty obrazovek pro výměny URCI-ZAKAZNIKA a ZMEN-DETAILY jsou zobrazeny na straně 102.

Na první obrazovce je jediný vstupní údaj začínající na řádku se textem "Cislo zakaznika :". První znak tohoto údaje je znázorněn podčárníkem (_). Při tvorbě tohoto formátu nejprve oznámíme proceduře pracující se slovníkem dat, že chceme vytvořit nový formát. Pak na prázdnou obrazovku natypujeme námi navrhovaný formát a to i s podčárníkem, určujícím začátek vstupního pole. Formát odešleme a systém se nás sám bude ptát na jméno údaje začínajícího za textem "Cislo zakaznika :". Odpovíme názvem tohoto údaje, tj. CISLO-ZAKAZNIKA. Systém se bude dále dotazovat na způsob zobrazení údaje, validaci a způsob zobrazení případné chyby. Tyto dotazy lze ignorovat, systém pak použije standardních činností. Tím je tvorba



Strukturovaný diagram pro aplikaci aktualizace zákazníků





formátu ukončena. Podobně postupujeme při návrhu druhé obrazovky. Zde je jeden výstupní údaj za textem "Zakaznik :" označený "*" a pět modifikovatelných údajů označených "!". Po odeslání formátu se nás systém postupně ptá na všech šest výstupních a modifikovatelných údajů.

4.4 Specifikace aplikačního procesu

Do slovníku dat vložíme elementy dialogů (DIALOGUE) a výměn (EXCHANGE), s náležitými parametry. Níže je přesný zápis těchto elementů do slovníku dat. Při skutečném zápisu do slovníku dat se obvykle namísto dlouhých názvů typů elementů a parametrů používají tříznakové zkratky.

```
INSERT DIALOGUE AKTUALIZUJ-ZAKAZNIKY
*SUBSCHEMA AMSUBSCH
*TASK      AMPROGRAM
*PROCESSING-STRUCTURE
   DIALOGUE AKTUALIZUJ-1-ZAKAZNIKA
   OCCURS UNTIL CISLO-ZAKAZNIKA = "DOST"
   AFTER URCI-ZAKAZNIKA OUTPUT-INTUP
**
INSERT DIALOGUE AKTUALIZUJ-1-ZAKAZNIKA
*PROCESSING-STRUCTURE
   EXCHANGE URCI-ZAKAZNIKA
   EXCHANGE ZMEN-DETAILY
**
INSERT EXCHANGE URCI-ZAKAZNIKA
*DISPLAY-STRUCTURE URCIZAK
*SINGULAR-VIEW     VETA-ZAKAZNIKA
*SELECTION         VETA-ZAKAZNIKA
                   WHERE MATCHINS CISLO-ZAKAZNIKA
*FAIL              IF FAILURE ON VIEW-RETRIEVAL
                   WITH MESSAGE "Takovy zakaznik neexistuje"
**
INSERT EXCHANGE ZMEN-DETAILY
*DISPLAY-STRUCTURE ZMENDET
*SINGULAR-VIEW     VETA-ZAKAZNIKA
                   SAME AS IN URCI-ZAKAZNIKA
*CHANGES           VETA-ZAKAZNIKA UPDATED
**
```

Povšimněme si, že v popisu neexistují žádné přesuny dat z databáze na obrazovku a naopak. Jsou-li názvy údajů v databázi i na obrazovce stejné, systém zajistí přesuny sám.

5. Závěr

Cílem tohoto příspěvku mělo být seznámení se zajímavým produktem firmy ICL. Je to neobyčejně silný nástroj pro tvorbu terminálově založených aplikací, který přitom ponechává programátorovi dost tvůrčího prostoru. Programování a údržba jsou velmi rychlé. Považujeme-li každý dialog nebo výměnu za modul, lze za jeden den napsat několik desítek modulů. Výsledný program je zvláště v oblasti vstupu a výstupu obrazovky velmi dobře propracovaný. Vysoká rychlost programování je dána předdefinováním většiny funkcí, takže programátor se stará spíše o výjimky, standardní části dělá systém sám. Zajímavý je také způsob primitivního a přitom účinného řešení strukturovaného přístupu.