SYSTÉMOVÁ PODPORA KOMUNIKAČNÍCH ÚLOH

Z. Rusín

1. Úvod

Otázky efektivnosti a racionality tvorby aplikačního programového vybavení v československých podmínkách jsou kladeny již dlouho. Všeobecná úroveň odpovědí však dosud nepřekročila lokální rámec několika příbuzných instalací, což ve chvíli ohlašovaného masového nástupu interaktivních zařízení do různých sfér našeho hospodářského života dává vysokou pravděpodobnost vzrůstajícího rozporu mezi uživatelskými potřebami a reálnými možnostmi geograficky i organisačně roztříštěných projekčně - programátorských skupin.

Předkládaný článek obrací pozornost k jedné, podle názoru autora poměrně nejsnáze, protože centralisovaně, uskutečnitelné inovaci programátorsko - aplikačních prostředků, jakou může být zásadní změna úrovně systémové podpory interaktivních a komunikačních úloh.

Přestože vývoj dávno překonal původně nesymetrický vztah počítač - terminál, zůstává tato představa v aplikačním programování žádoucí, je-li výrazem praktického požadavku, aby se koncové zařízení programu jevilo jako každý jiný datový soubor. Tím je zároveň do značné míry vymezen soubor nároků na operační systém, jemuž jsou přenechány všechny činnosti od konektování zařízení přes polling, transformaci a vysílání či příjem dat až po analysu komunikačních chyb a zotavení po nich, samozřejmě s co nejmenším důsledkem pro stav aplikace, která musí být jednoznačně informována o stavu spojení.

Z tohoto pohledu je relativně triviální záležitostí rozšířit nesymetrický vztah terminálové sítě k centrálnímu počítači na rovnomocnou síť výpočetních systémů pomocí pojmu konceptuálního terminálu, jímž se jeví jeden výpočetní systém druhému, aniž by bylo nutno měnit vžitá programátorská hlediska. Systémová podpora musí samozřejmě vyřešit požadavek, aby se týž systém choval v síti

současně jako iniciátor přenosů vůči jedněm systémům a jako odpovídající strana vůči systémům jiným, přičemž toto chování je iniciováno uživatelskými požadavky a je veskrze dynamické povahy.

Vyjděme nyní z analysy typických interaktivních a komunikačních úloh za formulací zdůvodněných požadavků na systémovou podporu při tvorbě a provozování takovýchto aplikací.

2. Individuální interaktivní úlohy

Dávno známou, původně scela lokální, interaktivní úlohou je proces operátorské aktivity, interpretující a vykonávající řídící oprátorské povely v podstatě týmž způsobem, jakým jsou v dávkové úloze zpracovávány příkazy popisu prací /JCL/. Prostým zobecněním obou typů příkazů do jednotného řídícího jazyka systému /SCL, viz článek /1/ / můžeme konstruovat základní reusabilní uživatelsky orientovaný počítačový proces, o jehož konkrétní podobě v rámci jednotlivé úlohy rozhodnou až parametry této úlohy.

Z našeho pohledu určujícím je pak interaktivní charakter základního vstupního proudu řídících příkazů v úlohách operátorsko-programátorských nebo alespoň jednoho souboru vstupních dat v úlohách uživatelského typu. Děje-li se totiž dialog v programátorsko-operátorské úloze převážně na úrovni řídícího jazyka systému, je nezbytné požadovat v uživatelských aplikacích výhradně komunikaci koncového uživatele s aplikačním vybavením na úrovni odborného jazyka tohoto uživatele a je nikoli posledním úkolem tvůrců aplikačního vybavení systémovou vrstvu v aplikaci uživateli scela utajit při řádném ošetření všech výjimečných stavů.

Praktická hlediska implementace různých uživatelských prostředí pro různé aplikace samozřejmě vnášejí potřebu diferencovat výkonné procesy už při jejich inicialisaci. Trvalé spojení procesu s některým koncovým zařízením je však omezující a zásadně nežádoucí. Jeho nezbytnost vyplývá vždy jen z nízké úrovně samotné systémové podpory.

Pro aplikaci musí být lhostejné, zda pracuje s lokálním či vzdáleným zařízením a simulace interaktivních vstupů a výstupů diskovými soubory se rovněž nesmí promítnout do uživatelského kódování, protože věta diskového souboru může být vždy kopií řádku

či celé obrazovky. Uživatelské vybavení nesmí být závislé na fyzických adresách koncových zařízení a v případě potřeby rozlišení vlastností interaktivních zařízení /řádkový nebo stránkový režim/ je nutno mít k dispozici takové systémové procedury, jimiž program zjistí vlastnosti právě připojeného zařízení a podle nich řídí svou činnost /např. formátování zpráv/. Ve všech ostatních směrech musí být programu jedno s jakým typem souboru pracuje. Že je takový přístup k datům možný a jak jej implementovat popsal autor v článku /2/.

Dokonce i tehdy, je-li aplikace v počítačovém systému jedinečná, nemá smysl spojovat uživatelský kód s konkrétním zařízením. Jde o prostou obdobu samozřejmé skutečnosti, že dávková úloha se chová stejně ať jsou její řídící příkazy exekuovány z jakéhokoli souboru, který je jedním z parametrů úlohy, nikoli neměnným atributem. Další důsledky tohoto pojetí čtenář najde v článcích /1/, /2/ a /3/.

3. Transakční zpracování

Poněkud jiná situace nastává u úloh transakčního zpracování, kde aplikace v několika procesech obsluhuje řadu koncových zařízení, obvykle několika typů. I zde, a tím spíš, platí všechny výše uvedené poznámky o nezávislosti programu na koncovém zařízení, přesto však úspěch při tvorbě programového vybavení pro transakční zpracování je určován něčím jiným, totiž kvalitou řídícího systému tohoto zpracování /komunikační monitor/. Není-li tento řídící systém samozřejmou součástí základního vybavení, jsou interaktivní zařízení využitelná jen v individuálních úlohách.

Rozhodne-li se pak některá instalace pro vlastní vývoj transakčního systému, odrazí se výchozí nulový stav vždy na kvalitě uživatelského servisu. Základním problémem programování se totiž stává nikoli otázka "co" programovat, nýbrž "jak" úlohy implementovat. A jak to souvisí s termíny realisace není nutno vysvětlovat. Uchráníme už přitom od nebezpečí, že řídící systém bude implementován aplikačními programátory na uživatelské a nikoli na systémové úrovni, což se může projevit na výkonnosti nejen samotné aplikace ale i celé instalace. Další stejně zásadní otázkou je výběr implementačního jazyka. Je-li systémová podpora omezena na soubor

makroinstrukcí v assemblerovém jazyce, připojují se všechny známé neduhy assemblerového kódování - nízká úroveň programátorských nástrojů, nízká produktivita ladění, malá čitelnost zdrojových textů atd.

Specifickou otázkou je tvorba obrazovkových formátů a způsob jejich začlenění v aplikačním kódu. Nejvýhodnější podobou je patrně forma textových modulů separovaných od ostatního kódu, přičemž vzorem může být před časem úspěšně užitá idea číselníkových modulů. Příslušné programátorské nástroje mají být samozřejmě součástí základního vybavení a měly by využívat všech známých technických možností obrazovkových zařízení.

4. Aplikace v počítačové síti

Dosud diskutované typy úloh nepřesahovaly rámec původního uspořádání počítač - terminál. Má-li být aplikace přístupná z jiného výpočetního systému v rámci počítačové sítě, přistupují do systémového vybavení nutně jisté zprostředkující složky, jež by zásadně opět měly být součástí základního vybavení.

Umí-li náš výpočetní systém uspokojit požadavek na individuální uživatelskou interaktivní úlohu, obslouží bez potíží i koncové zařízení kteréhokoli vzdáleného systému v síti. Samozřejmě za předpokladu, že tento vzdálený systém dokáže o takový servis požádat podle zásad komunikačního protokolu v síti a dokáže-li po navázání spojení být uspokojivým /pro aplikaci neviditelným/ prostředníkem mezi hostitelským systémem a daným koncovým zařízením.

V případě transakčního zpracování může být obdobná situace, kdy zařízení jednoho systému má být konektováno k procesům transakčního zpracování v jiném uzlu počítačové sítě, ale v úvahu přichází i jiná varianta, kdy zprávy si vyměňují dva transakční systémy ve dvou uzlech sítě. Připomeneme-li si skutečnost, že kterýkoli systém v síti má umět být stranou vyzývající i stranou odpovídající, může nám síť degenerovat v jediný systém a výměna zpráv mezi několika nezávislými transakčními procesy zůstává naprosto smysluplným požadavkem.

Je-li uvažovaná síť homogenní v tom smyslu, že ji tvoří instalace téhož typu, je na místě požadavek, aby aplikační programo-

vání bylo od problémů spojení v síti izolováno dostatečně mocnou systémovou podporou na úrovni vyšších programovacích jazyků. Vyskytnou se však situace, kdy má být realisováno spojení různorodých výpočetních systémů, což standardní vybavení ne vždy zcela vyřeší. Pak je potřeba mít k dispozici prostředky pro výměnu dat mezi aplikacemi v síti, přičemž aplikační programy přebírají zodpovědnost za nižší logické funkce vícevrstvého ISO modelu sítí s otevřenými systémy /OSI, viz výklad v článcích /4/, /5/, /7/ a publikaci /6/ /.

Z tohoto implementačního pohledu se stává nepodstatnou skutečnost, zda jde o úlohu interaktivního či dávkového charakteru, což je hledisko lokální, a do popředí vystupují otázky komunikace v síti. Zdá-li se čtenáři toto tvrzení přehnané, nechť promyslí problém výměny datových souborů mezi uzly sítě. Jde o úlohu ryze dávkovou, totiž o prosté zobecnění spoolingu. Na rozdíl od všech dosud diskutovaných aplikačních možností zde realisaci podstatně ovlivňuje přenosová rychlost spojení a charakter užitého komunikačního protokolu.

Používá-li se protokol znakově orientovaný, jak bylo v terminálových sítích zvykem, přináší nepříjemné komplikace už triviální požadavek na přenos binárních dat. Řešení, které zobrazuje jeden byte pomocí znakové representace dvou hexadecimálních číslic, vede k zdvojnásobení objemu přenášených dat. Základní otázkou je restartovatelnost přenosu s minimální ztrátou již odvysílaných dat, což vyžaduje mimořádně precisní kód ošetřující chybové stavy spojení a zvýšené nároky na kvalitu přenosových cest.

Předchozí výklad přinesl řadu požadavků na systémovou podporu komunikačních a interaktivních procesů a je na čase se pokusit o nástin koncepce, usilující o co největší nezávislost aplikačního kódu na koncových zařízeních, integrující veškeré projekční a programovací činnosti a vykazující patřičnou produktivitu ve fázi vývoje i při provozování aplikací.

5. Systémový katalog, systémové a uživatelské procesy

Požadujeme nezávislost uživatelského vybavení na konkrétních koncových zařízeních. Protože koncové zařízení je v programu re-

prezentováno datovým souborem, lze problém sobecnit na oddělení popisu vlastností datového souboru od uživatelského programu. Tato otázka byla zkoumána v článku /2/ a řešením je povýšení funkce adresáře souborů na systémový katalog, zavádějící navíc vlastnické a přístupové relace mezi svými objekty, jimiž kromě objektů uživatelské sféry - instance uživatele, magnetického media, datového souboru či knihovny souborů - jsou i hardwareová zařízení a konceptuální systémové entity.

Každý objekt je jednoznačně určen svým typem a v rámci systému jedinečným jménem, jež je přirozeným prostředkem parametrisace systémových i uživatelských rutin. Řady katalogových objektů tvaru {popis reusabilního procesu - prototypový popis uživatelského servisu - konkrétní uživatelský servis - přístupový uzel servisů v síti} a {prototypový popis technického zařízení - reálné koncové zařízení nebo konceptuální komunikační jednotka - přístupový uzel servisů v síti} se setkávají v komplexní entitě, spojující reálnou hierarchii komunikačních zařízení s řadou možných reálných i konceptuálních koncových zařízení na jedné straně a s řadou uživatelských služeb, jež mohou být v daném uzlu sítě uspokojovány, na straně druhé. Logickým dovršením je geografické členění, přiřazující hardwareovým zařízením katalogované místo, jímž lze z uživatelského hlediska určovat například reálné místo tiskových výstupů /při eventuální implicitní výměně datových souborů mezi uzly sítě/ a se systémového hlediska třeba členit pravomoci mezi řadu operátorských stanovišť obsluhujících periferie v jednom místě distribuované instalace.

Pro variabilní sítě, kde jediná modemová konekce v průběhu časové jednotky obsluhuje řadu vzdálených zařízení, může být žádoucí možnost dočasného katalogování momentálních koncových zařízení, která je včetně generování jejich jmen přenechána speciálnímu systémovému procesu ustavujícímu vnější spojení. Pak přihlašující se uzel musí v rámci navazování spojení dodávat dostatečnou informaci o svých vlastnostech, viz opět /4/, /6/, /7/.

Další systémový proces nechť je určen k selekci servisu koncovými uživateli v rámci inicialisace interaktivní úlohy. Ze znalosti stavu servisů v systému a katalogové informace o koncovém zařízení tento proces konstruuje vstupní menu služeb, které mohou

být v daném čase uspokojeny a uživatel si vybere svůj servis. Takové menu může být hierarchisového podle typu služeb nebo podle míst jejich poskytování a mělo by obsahovat nápovědu a naopak možnost zkrácené selekce pro stálého uživatele. Lze využívat i akčních kláves podle typu zařízení.

Uživatelův požadavek je dále postoupen spolu s jeho zařízením řídícímu procesu daného servisu /např. řídícímu systému transakčního zpracování/. Tento proces validuje požadavek vůči přístupovým právům uživatele k servisu a vůči limitům určujících hodnot úlohy /např. kvantum procesorového času, množství reálné paměti, způsob realisace výstupů v individuální úloze/ a podle plánovacích pravidel a povahy servisu je uživatelský požadavek buďto ihned obsloužen nebo akceptován a zařazen do fronty úloh. Externí komunikací mezi procesy jsou vybavovány i požadavky na zařazení dávkových úloh a výstupů ze všech typů úloh do front příslušných servisů, tj. do front jejich řídících procesů.

Každý typ spracování má tedy vlastní řídící a plánovací proces a řadu reusabilních procesů výkonných, v nichž jsou realisovány uživatelské požadavky jako jednotlivé kroky opakujícího se cyklu. Je možné, aby jeden typ procesu poskytoval řadu příbuzných servisů. Katalogový popis výkonného procesu obsahuje údaje o procesorových a paměťových prioritách, jméno řídícího procesu a seznam svých inicialisačních procedur. V popisu řídícího procesu jsou navíc organisační údaje jako přiřazení front servisům, plánovací algoritmus, jméno diskového souboru obsahujícího fronty a další.

V případě operátersko - programátorské interaktivní úlohy poslední inicialisační procedura v reusabilním cyklu dotváří uživatelské prostředí a interpretuje a vykonává příkazy řídícího jazyka systému. U uživatelské individuální úlohy degenerují příkazy řídícího jazyka v implicitní vykonání jediného příkazu startujícího aplikaci. Tímto příkazem může ovšem být volána obecná menu procedura, která pro každého uživatele nabídne jiné konkrétní akce podle katalogových dat daného uživatele. Jde tedy o schéma nadmíru tvárné.

6. Další systémové prostředky

Systém musí dovolovat dvojí způsob přiřazování koncových zařízení. Pro lokální výstupní zařízení je obvyklé požadovat konkrétní jednotku jménem, u komunikací naopak řídící procesy "naslouchají" požadavkům koncových zařízení. Naslouchacím heslem je jméno servisu. Jsou-li některá zařízení vyhražena jednomu servisu, mělo by být možné jim naslouchací jméno přiřadit i v katalogu, čímž se vyloučí nutnost selekce servisu vůbec a impuls z koncového zařízení je směrován přímo k řídícímu procesu daného servisu. Tímto způsobem lze přiřadit periferie procesům vstupního spoolingu.

Přímou součástí operačního systému musí tedy být privilegované procedury pro manipulaci se systémovými katalogovými entitami jako je zavádění, modifikace a likvidace objektů a vztahů mezi nimi. Zatímco některé modifikace se mohou uplatnit okamžitě, jiné až po nové inicialisaci jimi zasažených procesů.

Pro specialisované profesionální tvůrce systémového vybavení musí existovat procedury jimiž se realisují naslouchací požadavky, přiřazování a uvolňování zařízení, předávání zařízení mezi procesy, komunikace mezi procesy a jejich synchronisace a samozřejmě i fysické řízení magnetických a nemagnetických souborů dat. Prostředky logického řízení souborů patří do základního vybavení, viz DAM v článku /2/. Jde o spojení datového souboru s technickým zařízením a jeho přiřazení aplikačnímu programu, ostatní činnosti mají být rutinnímu programátorovi zprostředkovány základním vybavením. V zásadě všechny tyto procedury musí být přístupné z vyšších jazyků.

Operační systém má obsahovat úplný kód inicialisačních procedur řídících i výkonných procesů. Tyto mohou být podle potřeb konkrétní instalace doplňovány instrumentačními uživatelskými rutinami pro evidenci úloh a spotřeby zdrojů, modifikaci plánovacích algoritmů, dotváření uživatelského prostředí a podobně.

Řídící procesy jsou po inicialisaci suspendovány a očekávají na dva typy externí komunikace s jinými procesy - na zásahy do plánovacích algoritmů a na zprávy z procesů jimi řízených. Mezi prvý typ komunikace můžeme zahrnout i požadavky na poskytnutí servisu či zařazení úloh do front. Výkonný proces po inicialisaci rovněž

zůstává v klidu v očekávání řídících impulsů a po realisaci uživatelského požadavku se do klidového stavu opět vrací.

Celý systém transakčního zpracování může být uspořádán přesně týmž způsobem a tvorba aplikace představuje kompletaci obecných procesů uživatelskými rutinami.

Inicialisace a ovládání řídících procesů se má dít privilegovanými příkazy řídícího jazyka systému, jež jsou přístupné jen z patřičných operátorských procesů a podléhají restrikcím podle přístupových práv konkrétních katalogovaných operátorských uživatelů k daným servisům a procesům, ale též podle geografických míst jejich operování.

Protože smyslem článku není a jistě ani nemůže být detailní návod k implementaci takového v mnoha významech distribuovaného operačního systému, obraťme pozornost k otázkám integrovaného procesu vývoje úloh transakčního charakteru, čímž navážeme na článek /3/, v němž byly diskutovány otázky obecné podpory aplikačního programování.

7. Slovník dat

Bylo řečeno, že smyslem tvorby aplikačního vybavení pro úlohy transakčního zpracování není tvorba řídícího systému, nýbrž vybavení obecného mechanismu uživatelským kódem využívajícím všech technických možností terminálů i funkčních možností operačního systému /databázové prostředky s ochranou souborů, automatisace činností řízených obsahy katalogových uzlů atd./. K tomu má sloužit integrovaný komplex projekčně - programátorských prostředků.

Jednotící úlohu v procesu tvorby aplikačního programového vybavení může sehrávat slovník dat, jehož funkci rozšíříme na centrální bázi vědomostí o aplikační sféře týmž způsobem, jakým jsme v předchozích odstavcích rozšířili adresář souborů na systémový katalog.

Takovýto slovník dat lze realisovat, ostatně stejně jako katalog, v podobě hierarchické databáse /např. IDMS/, jejíž řídící informace /directory/ může být dále modifikovatelná podle potřeb konkrétní instalace.

Slovník je využíván v předprojektové přípravě pro popis reálných procesů, materiálových a informačních toků mezi nimi, v projektové fázi pro popis informačních procesů, jejich řídících a datových struktur, pro formální specifikace uživatelských programů a procedur ale též pro návrh všech formulářů /budou-li zapotřebí/ a obrazovkových formátů a samozřejmě je výchozím /a možná i jediným/ závazným "dokumentem" pro fázi programování. Čtenář nechť tento výčet srovná s počátečními možnostmi DDS firmy ICL z poloviny sedmdesátých let v publikaci /8/.

Pro každou tuto funkci má existovat specialisované uživatelské rozhraní s prostředky interaktivního vytváření a modifikování slovníkových informací a interaktivního či dávkového zpracování formátovaných výpisů. Pro zavádění popisů již existujících dat nemají chybět procesory databázových schemat a subschemat i popisů konvenčních souborů z uživatelských zdrojových programů. Všechny tyto prostředky jsou zcela běžným vybavením. Zastavit se však zde, by znamenalo nevyužít té nejdůležitější možnosti takovéto "dictionary", jíž bezesporu je schopnost rozšíření specifikačního jazyka na popisy obrazovkových formátů a řídících informací transakčního zpracování - výčet typů zpráv a programů je ošetřujících, přiřazení těchto programů výkonným procesům, popis periodických funkcí a specifikace, ostatně možná ne nezbytná, konektovatelných koncových zařízení.

V programátorském vybavení pak musí být procesory generující přímo cílovou podobu modulů s řídícími strukturami a obrazovkovými formáty transakčního zpracování. Zřejmě však není jediného důvodu nepokračovat i nyní dále. Nechť existují i generátory databázových řídících struktur a uživatelských procedur z vpodstatě neproceduálních specifikací v dictionary. Proč by toto neměla být nejproduktivnější cesta tvorby aplikačního vybavení vůbec? /viz ostatně nynější podobu DDS v manuálu /9/ /.

Takový specifikační jazyk nemusí být zdaleka všeobsáhlý. Nestandardní situace lze vyřešit formou externích procedur zasouvaných do strukturované stavby automatisovaně generovaného kódu. Úspory v programování i ladění jsou zřejmé. Kolem jednotlivých procesorů je dále možno vytvořit organisačně - evidenční nádstavbu, takže lze automatisovaně vést i značnou část průběžné

agendy o stavu projektu. Na druhé straně nelze zastírat, že jde o dalekosáhlý zásah do formální i faktické dělby práce mezi projekčními a programátorskými složkami, a bude-li takový systém v dohledné době třeba v systémech JSEP II k dispozici, zde jistě leží jedno z úskalí při jeho zavádění do praxe.

8. Závěr

Autor se pokusil podat koncepci výpočetního systému, tvořícího jeden uzel počítačové komunikační sítě, jako souhrnu systémových a uživatelských činností distribuovaných mezi řadu koexistujících spolu komunikujících procesů, jež jsou řízeny jednak uživatelskými požadavky, jednak uživatelsky modifikovatelnými informacemi o systémových objektech v databázové struktuře systémového katalogu. Standardní funkce všech procesů tak mohou být doplňovány uživatelským kódem, který se může vyvíjet v souladu s potřebami konkrétní instalace.

Procesy transakčního zpracování jsou přímou analogií uvedených systémových principů, kde řídící informace definuje konkrétní aplikační náplň obecného transakčního schematu.

V paralele k centrální funkci katalogu v systémové oblasti byla vytýčena obdobná úloha v aplikační sféře slovníku dat, jehož projekčně - progamátorská nadstavba byla rozšířena na možnost formalisované specifikace všech složek aplikačních produktů s navazujícím přímým generováním cílového kódu aplikačního vybavení.

Ve shodě s uživatelskou instrumentací systémově zaváděných procesů byla postulována táž možnost v kterémkoli místě automaticky vytvářeného aplikačního kódu.

Autor se nesnaží čtenáře přesvědčit o téměř transcendentální universalitě tohoto pojetí, necítí se ani dostatečně kvalifikovaný k prokázání jeho pokrokovosti z hlediska současného světového poznání v oboru, má však zato, že ve srovnání s převládající realitou u nás provozovaných výpočetních systémů by asi i v příštím desetiletí uvedená koncepce neobstála nikterak špatně.

Literatura :

1. Rusín Z., Výpočetní systéry příštích let a jejich dopad na profesní sféru, sborník Programování 82
2. Rusín Z., Užití databázových přístupů v řízení dávkového zpracování úloh HZD a ve VTV aplikacích, sborník Programování 83
3. Rusín Z., Kompilační, testovací a diagnostické prostředky v interaktivním prostředí, sborník Programování 84
4. Sokol J., Saudek J., X.25, protokol pro veřejné datové sítě, sborník SOFSEM 82
5. Pištělák J., Lokální počítačové sítě ..., sborník SOFSEM 82
6. Guide to IPA, TP100C0, ICL, London 1982
7. Kemp J., Reynolds R., The ICL information processing architecture IPA, ICL Technical Journal, vol 2, iss 2, ICL, London, November 1980
8. DDS Data Dictionary System, Technical Overview, P1155, ICL, London 1977
9. Data Dictionary System DDS.650, R0120/00, ICL, London 1983