PŘIHLÁŠKA VYNÁLEZU

zveřejněná podle § 31 zákona č. 527/1990 Sb.

(21) Číslo dokumentu:
2000 - 4827

(13) Druh dokumentu: **A3**

(51) Int. Cl.[7]:
G 06 F 19/00
G 06 F 17/30
G 06 F 15/163

(19) ČESKÁ REPUBLIKA

ÚŘAD PRŮMYSLOVÉHO VLASTNICTVÍ

(22) Přihlášeno: **21.12.2000**

(40) Datum zveřejnění přihlášky vynálezu: **14.08.2002**
(Věstník č. 8/2002)

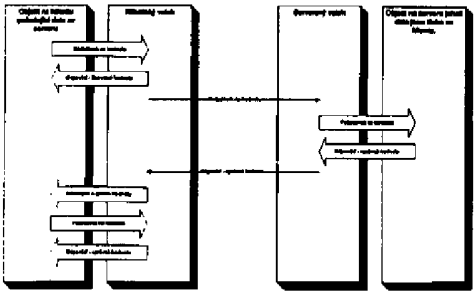
(71) Přihlašovatel:
HEAVEN INDUSTRIES, S. R. O., Praha, CZ;

(72) Původce:
Brož Petr, Říčany u Prahy, CZ;

(74) Zástupce:
Reichel Pavel Ing., Lopatecká 14, Praha 4, 14700;

(54) Název přihlášky vynálezu:

Způsob udržování logické integrity mezi objekty v interaktivních aplikacích počítačů, jeho použití pro systém sdílení dat v sítích, systém pro sdílení dat v sítích, způsob sdílení dat v sítích a použití tohoto způsobu



(57) Anotace:

Logické vztahy mezi jednotlivými objekty se pouze deklarují vazbami mezi jednotlivými objekty aplikace tak, že se vybírají vztahy ze souboru obecných, předem definovaných základních vztahů a jednotlivé objekty se připojují na výstupy nebo vstupy daného vztahu, načež každému základnímu vztahu se přiřadí obecný předem definovaný prostředek pro naplňování a udržování těchto vztahů při jejich používání. Objekty informují tyto prostředky, přiřazené ke vztahům, které na jejich hodnotách závisejí, o změnách svých hodnot, a prostředky reagují na změny v hodnotách svých vstupů a za účelem zachování integrity vztahů mění své výstupy a pracují konkurenčně, kdy několik prostředků může být aktivních současně, takže při běhu aplikace se současně zpracovává vstup uživatele a provádí se udržování integrity vztahů.

CZ 2000 - 4827 A3

Způsob udržování logické integrity mezi objekty v interaktivních aplikacích počítačů, jeho použití pro systém sdílení dat v sítích, systém pro sdílení dat v sítích, způsob sdílení dat v sítích a použití tohoto způsobu.

Oblast techniky

Vynález se týká způsobu udržování logické integrity mezi objekty v interaktivních aplikacích počítačů, jeho použití pro systém sdílení dat v sítích, dále systému pro sdílení dat v sítích, způsob sdílení dat v sítích v tomto systému a použití tohoto způsobu sdílení dat.

Dosavadní stav techniky

Interaktivní, zejména databázové aplikace, se skládají z množství objektů, například tabulek, jejich sloupců, komponent uživatelského rozhraní a buněk, které jsou vzájemně vázány logickými vztahy. Aby logické vztahy v dané aplikaci byly platné, vytvářejí se pro každý takový vztah v aplikaci algoritmy pro jeho naplnění (etablování, zavedení) a následné udržování (zachování integrity), zejména při změnách v hodnotách objektů. Příklad logických vztahů mezi objekty v aplikaci může být následující:

- Mezi komponentou uživatelského rozhraní a sloupcem tabulky může existovat logický vztah „zobrazení hodnoty sloupce", to je daná komponenta uživatelské rozhraní zobrazuje hodnoty daného sloupce (znaky, čísla).
- Mezi dvěma tabulkami může být vztah „výběr z tabulky dle kritéria", tj. jedna tabulka obsahuje výběr z druhé tabulky na základě daného kriteria.
- Mezi buňkou a sloupcem tabulky může existovat vztah „součet sloupce", tj. daná buňka obsahuje součet hodnot sloupce ze všech řádek dané tabulky.

Nevýhodou současného stavu je skutečnost, že je velmi pracné vytvářet udržovací algoritmy a rovněž provádět jakékoli jejich změny. V některých případech je z hlediska využití procesoru (CPU) a pracnosti vývoje udržovacích algoritmů nákladné průběžně zachovávat integritu ve vztazích mezi jednotlivými objekty. V mnoha případech je levnější i přijatelnější, aby tato integrita dočasně zachovávána nebyla. Týká se to například aplikací, které pracují s velkým množstvím dat, vztahů, případně i uživatelů. V takových aplikacích často není ani možné, z hlediska výkonu počítače a paměťové kapacity, vztahy mezi objekty naplnit všechny a stále je

udržovat. V takovém případě je zapotřebí volit, ve kterých časových okamžicích je nutné tuto integritu obnovit. Mohou přitom vzniknout následující situace. Buďto jsou udržovací algoritmy vyvolávány příliš často, což zbytečně zatěžuje procesor počítače a prodlužuje odezvu počítače k uživateli, anebo se vyvolávají nedostatečně často a uživateli jsou v takovém případě prezentovány nekonzistentní údaje, to je údaje, které neodpovídají v daném okamžiku logickým vztahům v aplikaci.

Částečným řešením problému pracnosti při vytváření algoritmů pro naplňování a udržování logických vztahů v interaktivních aplikacích jsou tak zvané tabulkové procesory (spreadsheet). V těchto aplikačních prostředích jsou logické vztahy pouze deklarovány. Jejich naplňování a údržbu provádí automaticky na základě těchto deklarací sám tabulkový procesor. Tabulkový procesor pracuje takto:

- Pamatuje si u každého objektu seznam všech objektů, které na jeho hodnotě závisejí.
- Udržuje si seznam objektů, které je třeba přepočítat. Tento seznam je po prvotním naplnění vztahů prázdný.
- Pokud uživatel změní hodnotu nějakého objektu, zanesou se do seznamu objektů k přepočtení všechny objekty, které na daném změněné hodnotě závisí.
- Poté tabulkový procesor prochází seznam objektů k přepočtení a jeden po druhém přepočítává.
- Po přepočtení daného objektu jej ze seznamu k přepočtení vyřadí.
- Pokud při přepočítání dojde k nové hodnotě, která je různá od hodnoty před přepočtením, zařadí nově do seznamu všechny objekty, které na dané hodnotě závisejí.
- Toto provádí tabulkový procesor tak dlouho, dokud není seznam objektů k přepočtení prázdný.

Problémy s touto architekturou jsou následující:

1. Uživatel může měnit hodnoty (provádět interakci se systémem) jenom v období mezi dvěma přepočty. Pokud tabulkový procesor právě provádí časově náročný přepočet nějakého objektu, jako třeba součet sloupce veliké tabulky, musí uživatel čekat než tento přepočet skončí nebo musí být tento přepočet ukončen a později započat celý znovu. To vede buď ke zdržování uživatele nebo k plýtvání

výpočetním výkonem, které může vést až tak daleko, že se daný objekt nemusí dopočíst nikdy.

2. Jestliže aplikace pracuje s velkým množstvím dat a vztahů, pak plné integrity prakticky nelze dosáhnout nikdy. I pokud lze integrity dosáhnout, mohou výpočty trvat tak dlouho, že práci s aplikací prakticky znemožňují. Z tohoto důvodu se také tabulkové procesory pro interaktivní aplikace s větším množstvím dat a vztahů nepoužívají.

Podstata vynálezu

Nevýhody současného stavu jsou do značné míry odstraněny řešením podle tohoto vynálezu. Podstata tohoto řešení způsobu udržování logické integrity mezi objekty v interaktivních aplikacích počítačů spočívá v tom, že logické vztahy mezi jednotlivými objekty se pouze deklarují vazbami mezi jednotlivými objekty aplikace tak, že se vybírají vztahy ze souboru obecných, předem definovaných základních vztahů a jednotlivé objekty se připojují na výstupy nebo vstupy daného vztahu, načež každému základnímu vztahu se přiřadí obecný předem definovaný prostředek pro naplňování a udržování těchto vztahů při jejich použití, kde objekty informují prostředky, přiřazené ke vztahům, které na jejich hodnotách závisejí, o změnách svých hodnot, a prostředky reagují na změny v hodnotách svých vstupů a za účelem zachování integrity vztahů mění své výstupy a pracují konkurenčně, kdy několik prostředků může být aktivních současně, takže při běhu aplikace se současně zpracovává vstup uživatele a provádí se udržování integrity vztahů.

Uvedeným řešením se dosáhne toho, že je zajištěna rychlá odezva aplikace pro uživatele i při velkém množství dat a vztahů a zároveň nedochází k takovému plýtvání výpočetním výkonem jako v současných tabulkových procesorech. Navíc provádění jednoho časově náročného výpočtu neblokuje ostatní výpočty, ty probíhají současně. Uživatel tedy vidí výsledky nenáročných výpočtů „včas", přestože mohou nadále probíhat další náročné výpočty.

Při konkurenčním zpracování prostředků a vstupu uživatele se zpracování prostředků, tvořených algoritmy, s výhodou rozdělí na krátké časové úseky, mezi kterými může být algoritmus přerušen a během nichž přerušen být nemůže a okolní data se tedy nemění, a aplikace se řídí centrální smyčkou, určující okamžik, kdy

probíhá zpracování vstupů od uživatele a kdy se provádějí kroky uvedených algoritmů a kterých, přičemž jednotlivé akce se řadí vždy časově za sebou.
Při konkurenčním zpracování algoritmů může být několik algoritmů aktivních současně. Ke změnám ve vstupu daného algoritmu tak může dojít v průběhu jeho provádění. Výstupy vztahu existují, i když algoritmus ještě neskončil, a jsou případně měněny tak, jak pokračuje zpracování algoritmu.
Běh aplikace je složen ze dvou typů akcí, které probíhají konkurenčně:

1. zpracování vstupu uživatele
2. provádění udržovacích algoritmů

Zpracování vstupu uživatele může vést ke změnám v primárních datech. V takovém případě jsou o těchto změnách informovány všechny vztahy, které na hodnotách daných primárních dat závisí.

Při informování prostředků o změnách v okolí se pro výpočet nové hodnoty využijí před změnami vypočítané hodnoty. Lze tak dosáhnout výrazných úspor ve výkonu procesoru.

Prostředky pro naplňování a udržování logických vztahů mezi jednotlivými objekty se řídí poptávkou po hodnotách objektu tak, že naplňování a udržování vztahu probíhá jen při poptávce po výsledku, a po ukončení poptávky se naplňování nebo udržování ukončí. Jestliže jsou algoritmy řízeny poptávkou po hodnotách objektu, dosahuje se tím výrazné úspory výkonu procesoru i paměťové kapacity počítače. Poptávka primárně vychází z požadavku uživatele zobrazit hodnotu objektů, sekundárně pak poptávka vzniká tím, že pro naplnění hodnoty jednoho objektu (poptávaného objektu) je současně nutné znát hodnotu jiného objektu. Po ukončení poptávky s naplňování nebo udržování ukončí okamžitě nebo v časovém odstupu.

Všechny výstupy, mezivýsledky a stavové informace algoritmů se s výhodou ukládají do trvalé paměti. Tím lze dosáhnout podstatné úspory prostoru operační paměti. Jestliže se tak děje na konci každého kroku a poté algoritmus na začátku dalšího následujícího kroku všechny potřebné informace načte znovu, pak lze v libovolném okamžiku (po libovolném kroku) aplikaci okamžitě ukončit a později její běh obnovit ve stavu, ve kterém byla ukončena.

Při konkurenčním zpracování udržovacích algoritmů a vstupu uživatele se do systému s výhodou zařadí garbage collector pro mazání neužitečných hodnot, kterým se na základě informací o volné paměti počítače a dalších informací iniciuje mazání neužitečných hodnot. Tím lze opět dosáhnout podstatné úspory paměťového prostoru.

Podstatou řešení podle tohoto vynálezu je dále použití výše popisovaného způsobu udržování logické integrity mezi objekty v interaktivních aplikacích počítačů pro sdílení dat v sítích, zahrnující server k centrálnímu ukládání sdílených dat a nejméně jeden klient, na kterém je provozována aplikace, která využívá sdílená data a zpracovává informace o změnách sdílených dat, kdy klienty jsou alespoň přechodně spojeny sítí, opatřený pro aplikaci pro přístup na sdílená data z klientu vyrovnávací pamětí. Klientské a serverové vztahy rozšiřují využití aplikační architektury s automatickou údržbou logické integrity s konkurenčním zpracováním pro provoz na více počítačích, propojených v síti.

Podstatou řešení podle tohoto vynálezu je rovněž systém pro sdílení dat v sítích, zahrnující server k centrálnímu uložení sdílených dat a nejméně jeden klient, na kterém je provozována aplikace, která využívá sdílená data a zpracovává informace o změnách sdílených dat, kdy klienty jsou alespoň přechodně spojeny sítí, opatřený pro aplikaci pro přístup na sdílená data z klientu vyrovnávací pamětí, který pro použití uvedeného způsobu udržování logické integrity v interaktivních aplikacích počítačů zahrnuje dále soubor obecných, předem definovaných základních vztahů speciální klientský vztah s nejméně jedním výstupem a žádným vstupem, přičemž všechna sdílená data v klientské aplikaci jsou reprezentována objektem s připojeným klientským vztahem, pracujícím jako vyrovnávací paměť. Soubor obecných, předem definovaných základních vztahů zahrnuje dále speciální serverový vztah s nejméně jedním vstupem a žádným výstupem, přičemž ke všem objektům, jejichž hodnoty jsou sdíleny, jsou připojeny serverové vztahy pro reagování na požadavky klientů.

Je výhodné, při porovnání se současným stavem vývoje a provozu síťových aplikací, že na jedné straně se používá obecný mechanismus, který nevyžaduje dodatečné náklady na vývoj pro převedení izolované aplikace na síťovou; na druhé straně pro komunikaci mezi počítači lze použít asynchronní protokol, který není tak citlivý na

kvalitu spojení mezi počítači a tedy bez výrazné újmy na funkčnosti aplikace ji lze provozovat i v méně kvalitních sítích, například v síti Internet, a zejména v bezdrátovém Internetu. Podstatnou výhodou řešení podle tohoto vynálezu je, že lze libovolně rozdělit každou aplikaci na část klientskou a serverovou tak, aby pro většinu interakcí koncového uživatele aplikace nebylo nutné komunikovat se serverem. Tím lze minimalizovat čas čekání uživatele na server.

Podstatou řešení podle tohoto vynálezu je dále způsob sdílení dat v uvedeném systému, spočívající v tom, že v případě, kdy aplikace požaduje hodnotu, která ještě není uložena ve vyrovnávací paměti, zašle se po síti požadavek o tuto hodnotu na server a zároveň se aplikaci vrátí libovolná hodnota zpět. Jakmile se ze serveru vrátí odpověď, pak pokud není shodná s hodnotou, která již dříve byla vrácena aplikaci, tak se aplikaci oznámí změna hodnoty a hodnota obdržená ze serveru se zapamatuje ve vyrovnávací paměti. Jakmile klient obdrží po síti od serveru informace o změně hodnoty, oznámí se tato změna aplikaci a z vyrovnávací paměti se hodnota vymaže nebo se označí za neplatnou tak, aby při dalším požadavku aplikace na tuto hodnotu došlo k novému požadavku na server, a jakmile aplikace na klientu požaduje změnit některou sdílenou hodnotu, tak se nová hodnota okamžitě uloží do vyrovnávací paměti, přičemž aplikace je o změně hodnoty okamžitě informována a zároveň se na server zašle požadavek na změnu této hodnoty. Dokud klient od serveru neobdrží potvrzení, že server jeho požadavek zpracoval, ignorují se veškeré zprávy ze serveru o změně dané hodnoty. Požadavky klientů na zaslání hodnot se přijímají po síti serverem, který hodnoty zasílá zpět, jakmile je sám zjistí a zároveň si pamatuje, kterým klientům které hodnoty odeslal. Jakmile se změní některé hodnoty na serveru, odesílá se tato informace ze serveru těm klientům, kterým se odeslala původní hodnota a pokud se těmito klienty původní hodnota znovu nevyžádá, potom další informace o změnách dané hodnoty od serveru se jim nadále nezasílají. Server přijímá od klientů požadavky na změnu své hodnoty, které se zpracovávají tak, že server danou hodnotu změní a poté odešle klientovi, od kterého požadavek obdržel, informaci o provedené změně a zároveň se tato informace o změně hodnoty odesílá i dalším klientům, kterým server dříve odeslal původní hodnotu.
Jestliže aplikační architektura s automatickým udržováním logické integrity s konkurenčním zpracováním používá poptávkové řízení, je výhodné, že data jsou zasílána jen klientům, kteří je fakticky potřebují a stejně tak i informace o změnách.

Tím jsou minimalizovány celkové komunikační náklady. Výhodná je rovněž i skutečnost, že o změnách v datech, iniciovaných jedním uživatelem, se s minimální prodlevou dozví ostatní uživatelé, kteří tato data poptávají.

Při přerušení spojení mezi klientem a serverem se požadavky a informace o změně hodnoty s výhodou ukládají do paměti v místě vzniku této zprávy a odesílají se při následném obnovení spojení. Jednotliví klienti, stejně jako server, mohou být přechodně ukončeni nebo odpojeni od sítě, aniž by se narušila konzistence celého systému klientů a serverů. Navíc může uživatel s klientem často bez výrazné ztráty funkčnosti nadále pracovat, i když je od serveru odpojen. Většina uživatelských interakcí nevyžaduje komunikaci se serverem.

Aplikace se u uvedeného způsobu sdílení dat v systému může rozdělit na větši počet úrovní než klient a server, v podstatě na libovolný počet úrovní pracování. Kromě uvedených dvou úrovní klient – server lze aplikaci provozovat například na čtyřech úrovních, mezi klienta a server lze v takovém případě vložit na straně klienta vrstvu zástupce, který soustřeďuje komunikaci se serverem pro více klientských počítačů a na straně serveru vložit vyrovnávací servery, které rozdělují zpracování aplikační logiky na více vyrovnávacích počítačů. Dochází k rozložení výkonu na více počítačů na straně serveru, na klientské straně se jedná o minimalizaci komunikačních nákladů.

Další podstata tohoto vynálezu spočívá v použití uvedeného způsobu sdílení dat pro aplikace, sdílející data z více než jednoho serveru. Toto použití lze v praxi s výhodou realizovat tak, že po doplnění zástupce pro potřebná data z jiné aplikace o klientský vztah s identifikací počítače, zpracovávajícího tuto aplikaci, a doplnění v ní o vztahy server k datům, které se dávají k dispozici k integraci s jinými aplikacemi, se v okamžiku deklarace klientského vztahu určí, na který server má klientský vztah směřovat své požadavky. Zpracování informací mezi integrovanými aplikacemi probíhá automaticky v reálném čase, s minimálními náklady na vývoj propojení.

Přehled obrázků na výkresech

Vynález bude blíže vysvětlen pomocí výkresů a následujícího popisu příkladů provedení. Na obrázku 1 je znázorněn příklad centrální smyčky, která řídí aplikaci a

určuje, ve který okamžik se provádějí kroky algoritmů a kterých algoritmů. Výběr algoritmu k provedení může například probíhat tak, že se cyklicky prochází všechny aktivní algoritmy, to je ty algoritmy, které jsou v daném okamžiku v běhu.
Na obrázku 2, 3 a 4 jsou znázorněny tři typy kroků, na které je rozdělen algoritmus, který splňuje následující vlastnosti: informuje o změnách, zpracovává změny, pracuje krokově a je řízen poptávkou. Na obr.2 je znázorněn krok zjištění hodnoty sloupce S v řádku R, na obr.3 reakce na změnu hodnoty buňky B a na obr.4 postupná změna hodnot sloupce S. Na obr.5 je znázorněn příklad průběhu zpracování požadavku na hodnotu ze serveru při použití aplikační architektury s automatickým udržováním logické integrity s konkurenčním zpracováním v aplikacích, sdílených větším množstvím uživatelů, např.Internet. Na obr.6 je znázorněn příklad průběhu zpracování informace o změně hodnoty na serveru, na obr.7 příklad průběhu zpracování požadavku na změnu hodnoty na serveru.

Příklady provedení vynálezu

Logické vztahy mezi jednotlivými objekty v interaktivních aplikacích počítačů (například tabulkami, jejich sloupci, komponentami uživatelského rozhraní a buňkami) se pouze deklarují a jejich naplňování a udržování se provádějí automaticky a konkurenčně. Logické vztahy se deklarují vazbami mezi jednotlivými objekty aplikace, přitom samotné vztahy jsou vybírány ze souboru obecných, předem definovaných základních vztahů (katalogu).

Příklady logických vztahů mezi objekty v aplikaci jsou následující:

- Mezi komponentou uživatelského rozhraní a sloupcem tabulky může existovat logický vztah „zobrazení hodnoty sloupce", to je daná komponenta uživatelského rozhraní zobrazuje hodnoty daného sloupce (znaky, čísla).
- Mezi dvěma tabulkami může být vztah „výběr z tabulky dle kritéria", to je jedna tabulka obsahuje výběr z druhé tabulky na základě daného kriteria.
- Mezi buňkou a sloupcem tabulky může existovat vztah „součet sloupce", to je daná buňka obsahuje součet hodnot sloupce ze všech řádek dané tabulky.

Každý vztah má jeden nebo více výstupů a může mít jeden nebo více vstupů. Je to názorně vidět na následujícím příkladu:

vztah	Vstupy	výstupy
zobrazení hodnoty sloupce	1. sloupec, jehož hodnoty se mají zobrazit	1. komponenta uživatelského rozhraní, která má hodnoty zobrazovat
výběr z tabulky dle kritéria	1. tabulka, ze které se provádí výběr 2. sloupec této tabulky, podle kterého se provádí výběr (pokud sloupec obsahuje hodnotu ANO, bude řádek zařazen do výběru) 3. sloupce této tabulky, které se mají přenést do výběru	1. tabulka s výběrem 2. sloupce této tabulky, kam budou přeneseny hodnoty z původní tabulky
součet sloupce	1. tabulka, jež bude sčítána 2. sloupec, jenž bude sčítán	1. buňka, která bude obsahovat součet

Deklarace vazby mezi danými objekty je pak realizována vybráním vztahu z katalogu a připojením jednotlivých objektů na výstupy či vstupy daného vztahu. Každý objekt může být připojen nejvýše na jeden výstup jednoho vztahu.

Každému základnímu vztahu je přiřazena sada obecných, předem definovaných algoritmů pro naplňování a udržování těchto vztahů v kontextu jejich použití (udržovacích algoritmů).

Algoritmy splňují následující podmínky:

a) Informují o změnách; to znamená, že informují objekty, které na hodnotách jejich výstupu závisejí, o změnách svých výstupů. To, že na nějakém objektu A závisí jiné objekty B, se objekt A dozví od objektů B. Buďto se to dozví v okamžiku vytvoření objektů B, to je tak zvaně staticky. Anebo se to dozví tak zvaně dynamicky, to je až v okamžiku, kdy si hodnotu objektu A přečtou objekty B a stanou se fakticky na této hodnotě závislé. Může existovat libovolná kombinace obou těchto přístupů.
K uvedeným změnám dochází zejména v případě, že algoritmus přepočítává hodnotu výstupu na základě informací o změnách v hodnotách svých vstupů.

b) Zpracovávají změny; Algoritmy jsou připraveny na změny v hodnotách svých vstupů a reagují na ně patřičným přepočítáním svých výstupů, tak aby integrita vztahu byla zachována. Informace o změnách vstupů dostanou algoritmy jednak od jiných algoritmů nebo od tak zvaných primárních dat, to je objektů, které nejsou připojeny k žádnému výstupu žádného vztahu. K změnám primárních dat dochází zejména na přímý pokyn uživatele.

c) Pracují konkurenčně; to je několik algoritmů může být aktivních současně. Ke změnám ve vstupu daného algoritmu tak může dojít v průběhu jeho provádění. Konkurenční zpracování udržovacích algoritmů a vstupu uživatele může být například zabezpečeno takto:

Algoritmy pracují krokově, v krátkých krocích; to znamená, že jejich běh (zpracování) je rozdělen na časové úseky, které jsou malé a mezi kterými může být algoritmus přerušen. Algoritmy přitom předpokládají, že v rámci každého kroku se okolní data nemění a pokud se změní po nějakém kroku, tak se o této změně algoritmus dozví nejpozději na začátku následujícího kroku. „Malé" znamená, že zpracování každého kroku i nad téměř libovolně velkými daty probíhá tak rychle, že není z hlediska uživatele zaznamenatelné. Například časová náročnost každého kroku je nezávislá na množství dat, nebo při množství dat *n* má časovou náročnost nejvýše *O (log n)*.

Aplikace je řízena centrální smyčkou (její příklad je vyobrazen na obr.1), která určuje, ve který okamžik probíhá zpracování vstupů od uživatele a ve který okamžik se provádějí kroky uvedených algoritmů a kterých algoritmů. Jednotlivé akce jsou vždy řazeny ze sebou, nikdy neprobíhají současně.

Výstupy vztahu existují, i když algoritmus ještě neskončil, a jsou případně měněny tak, jak pokračuje zpracování algoritmu. Pro lepší orientaci navazujících algoritmů je možno zavést a používat speciální hodnotu (konstantu), která indikuje, že výstup algoritmu ještě není kompletní.

Běh aplikace je složen ze dvou typů akcí, které probíhají konkurenčně:

1. Zpracování vstupu uživatele
2. Provádění udržovacích algoritmů

Zpracování vstupu uživatele může vést ke změnám v primárních datech. V takovém případě jsou o těchto změnách informovány všechny vztahy, které na hodnotách daných primárních dat závisí.

Uvedeným řešením se dosáhne toho, že je zajištěna rychlá odezva aplikace pro uživatele i při velkém množství dat a vztahů a zároveň nedochází k takovému plýtvání výpočetním výkonem jako v současných tabulkových procesorech. Navíc

provádění jednoho časově náročného výpočtu neblokuje ostatní výpočty, ty probíhají současně. Uživatel tedy vidí výsledky nenáročných výpočtů „včas", přestože mohou nadále probíhat další náročné výpočty.

Výrazných úspor ve výkonu procesoru lze dosáhnout tím, jestliže algoritmy, pokud dostanou informaci o změně v okolí, nezačnou postup výpočtu provádět od počátku znovu, ale danou změnu pouze zapracují do již vypočítaných hodnot.
Výrazné úspory výkonu procesoru i paměťové kapacity počítače se dále dosáhne, jestliže algoritmy jsou řízeny poptávkou po hodnotách objektu. To znamená, že naplňování a udržování vztahu zásadně neprobíhá, pokud není poptávka po výsledku. Poptávka primárně vychází z požadavku uživatele zobrazit hodnotu objektů, sekundárně poptávka vzniká tím, že pro naplnění hodnoty jednoho objektu (poptávaného objektu) je současně nutné znát hodnotu jiného objektu.

Například při poptávce na objekt „součet" je nutné znát hodnoty všech sčítanců. Po ukončení poptávky se naplňování nebo údržba ukončí okamžitě, případně v časovém odstupu.

Následuje příklad algoritmu splňujícího tyto vlastnosti: Informuje o změnách, zpracovává změny, pracuje krokově a je řízen poptávkou. Tento algoritmus patří vztahu mezi sloupcem tabulky a buňkou, který lze slovy popsat jako „sloupec S obsahuje hodnotu buňky B". Úkolem algoritmu tudíž je, aby hodnota sloupce S ve všech řádcích tabulky byla stejná a to rovna hodnotě v buňce B.
Problém je, že při změně hodnoty v buňce je třeba změnit hodnoty ve všech řádcích tabulky, což obecně nemůže být provedeno v jednom kroku, protože s dostatečně velkou tabulkou to může trvat libovolně dlouho.

Algoritmus je rozdělen na 3 typy kroků:

- zjištění hodnoty sloupce S v řádku R
- reakce na změnu hodnoty buňky B
- postupná změna hodnot sloupce S

První krok je vyvolán jako reakce na požadavek na hodnotu od okolních vztahů.
Druhý krok je vyvolán jako reakce na oznámení buňky B, že se její hodnota změnila.

Třetí krok je vyvolán krokem druhým (viz níže). Algoritmus ze svého okolí požaduje hodnotu buňky B.

Algoritmus si pro sebe ukládá do paměti následující hodnoty:

- číslo A aktuální verze hodnoty buňky B
- pole H aktuálních a historických hodnot buňky B indexované číslem verze
- pole V verzí hodnoty buňky B platných v jednotlivých řádcích sloupce S indexované číslem řádku
- frontu F všech platných řádků sloupce S, do které lze přidávat na konec a číst ze začátku

Jednotlivé kroky algoritmu jsou znázorněny dále graficky na obr.2, 3 a 4. Začátek a konec kroku je v kroužku, větvení kosočtvercem, provedení akce čtvercem. Zjištění hodnoty sloupce S v řádku R je graficky zobrazeno na obr.2, reakce na změnu hodnoty buňky B na obr.3 a postupná změna hodnot sloupce S na obr.4.

Podstatné úspory prostoru operační paměti lze dosáhnout tím, pokud algoritmy ukládají všechny své výstupy mezivýsledky a stavové informace do zvláštní paměti, např. na disk. Pokud se tak děje na konci každého kroku a poté algoritmus na začátku dalšího následujícího kroku všechny potřebné informace načte znovu, a pokud je zvláštní paměť trvalého charakteru (viz výše, např.disk), pak lze v libovolném okamžiku (po libovolném kroku) aplikaci okamžitě ukončit a později její běh obnovit ve stavu, ve kterém byla ukončena.

Podstatné úspory paměťového prostoru lze dosáhnout tím, že je do systému zařazen zvláštní algoritmus pro mazání neužitečných hodnot (garbage collector – GC), kterému jednotlivé algoritmy sdělují informace o tom, které hodnoty jsou neužitečné, a který na základě informací o volné paměti počítače a dalších informací iniciuje mazání těchto nepotřebných hodnot. Nepotřebnost a potřebnost hodnot se může řídit podobným principem jako je princip řízení poptávkou po hodnotách objektu, viz výše. Algoritmy, které spolupracují s GC, musí umožňovat smazání nepotřebných hodnot při zachování tří nutných podmínek, to jest pracují konkurenčně, sdělují a přijímají

změny. Pokud po smazání vznikne opětná potřeba těchto již smazaných hodnot, musí je tyto algoritmy znovu vygenerovat.

Internetové aplikace, sdílené velkým množstvím uživatelů, jsou v současné době v zásadě založeny na dvou různých architekturách.

První z nich je terminálová architektura, to je např. využívající HTML/HTTP, WML/WAP, kde všechno nebo téměř všechno zpracování probíhá na serveru a pro komunikaci se serverem se používá obecný protokol. Problémy s touto architekturou jsou:

- Při každé akci uživatele je třeba kontaktovat server pro její zpracování. To znamená, že uživatel ztrácí mnoho času čekáním na server. Jestliže se Internet používá bezdrátově nebo s méně kvalitním spojením, pak tyto problémy mohou uživatele odrazovat od používání aplikace.
- Mezi klientem a serverem se posílá zbytečně mnoho dat, které už terminál jednou načetl. Tím se opět prodlužuje odezva uživateli a zároveň se zahlcuje komunikační médium.
- Pokud ke změně v datech na stránce dojde bez vlivu uživatele (např. jiný uživatel, který užívá stejnou stránku, provede změny), není tato stránka automaticky obnovena a uživatel se o tom nemusí ani dozvědět. Druhá architektura spočívá v tom, že se provozuje samostatně část aplikace na straně koncového uživatele (na klientu) a část na serveru a pro komunikaci mezi nimi se používá protokol speciálně vyvinutý pro danou aplikaci. To s sebou nese tento problém:
- Vývoj speciálních protokolů a aplikací, které je využívají, je mimořádně nákladný pro středně velké a větší aplikace. Tato metoda se proto používá převážně v jednoduchých aplikacích, například pro e – mail.

Pro odstranění výše zmíněných problémů se použije výše popisovaná aplikační architektura s automatickým udržováním logické integrity s konkurenčním zpracováním následujícím způsobem.

- Soubor (katalog) předem definovaných základních vztahů obsahuje dvojici speciálních vztahů klient a server, popsaných níže. Vztah klient má jeden výstup a žádný vstup. Vztah server má jeden vstup a žádný výstup.

- Místo jedné izolované aplikace se vytvářejí aplikace dvě, klientská a serverová, s tím, že klientská bude využívat data aplikace serverové. Všechny serverové objekty, jejichž hodnoty jsou potřeba pro klientskou aplikaci, jsou v klientské aplikaci reprezentovány objektem s připojeným klientským vztahem. V serverové aplikaci jsou k těmto objektům připojeny serverové vztahy.
- Serverový vztah přijímá požadavky od klientů na zaslání své hodnoty přes síť. Tyto požadavky vyřizuje tak, že se zeptá k sobě přiřazeného objektu a výsledek opět po síti odesílá klientovi. Zároveň si pamatuje, kterým klientům odeslal hodnoty výsledku. Pokud od sobě přiřazeného objektu dostane informace o změně jeho hodnoty, pak tuto informaci rozesílá klientům, o kterých ví, že jim původní hodnotu odeslal. Pokud si tito klienti původní hodnotu nevyžádají znovu, další informace o změnách od serverového vztahu už nedostávají. Dále serverový vztah přijímá od klientů požadavky na změnu své hodnoty. To se týká pouze těch serverových vztahů, které jsou přiřazeny k primárním datům. Tento požadavek vyřizuje serverový vztah tak, že požadavek postoupí svému přiřazenému objektu a poté, co je tímto přiřazeným objektem zpracován, odešle klientovi, od kterého požadavek dostal, informaci o úspěšně provedené změně. Zároveň pošle informaci o změně hodnoty i ostatním klientům, kterým dříve původní hodnotu zaslal.
- Klientský vztah požadavky na svou hodnotu, pokud ji ještě nezná, vyřizuje tak, že zašle po síti požadavek o tuto hodnotu na server a zároveň okamžitě vrátí libovolnou hodnotu zpět. Když se klientskému vztahu vrátí odpověď ze serveru, nastanou dvě možnosti. Buďto je vrácená hodnota stejná s hodnotou, kterou již dříve vrátil; pak není potřeba žádné dodatečné akce. V opačném případě oznámí změnu své hodnoty všem na něm závislým objektům a hodnotu obdrženou ze serveru si zapamatuje. Pokud příště je požádán o svou hodnotu, vrátí již hodnotu, kterou si dříve zapamatoval. Příklad průběhu zpracování požadavku na hodnotu ze serveru je znázorněn na obr.5. Když klientský vztah obdrží po síti od serveru informaci o změně hodnoty, oznámí tuto změnu všem na něm závislým vztahům a nadále se chová tak, jakoby žádnou hodnotu od serveru neobdržel (neznal). Příklad průběhu zpracování informace o změně hodnoty na serveru je znázorněn na obr.6. Klientský vztah požadavky na změnu své hodnoty (pokud tento vztah zastupuje primární data) vyřizuje tak, že okamžitě změní svou hodnotu, informuje o této změně všechny na něm závislé objekty a zároveň

požadavek na změnu své hodnoty zašle na server. Dokud od serveru neobdrží potvrzení, že server jeho požadavek zpracoval, ignoruje veškeré zprávy ze serveru o změně dané hodnoty. Příklad průběhu zpracování požadavku na změnu hodnoty na serveru je znázorněn na obr.7.

Klientské a serverové vztahy rozšiřují využití aplikační architektury s automatickou údržbou logické integrity s konkurenčním zpracováním (AULIK) pro provoz na více počítačích, propojených v síti.
Další výhoda při porovnání se současným stavem vývoje a provozu síťových aplikací je v tom, že na jedné straně se používá obecný mechanismus, který nevyžaduje dodatečné náklady na vývoj pro převedení izolované aplikace na síťovou; na druhé straně pro komunikaci mezi počítači je použit asynchronní protokol, který není tak citlivý na kvalitu spojení mezi počítači, a tedy bez výrazné újmy na funkčnosti aplikace ji lze provozovat například i v méně kvalitních sítích, například v síti Internet, a zejména v bezdrátovém Internetu. Podstatnou výhodou řešení je, že lze libovolně rozdělit každou aplikaci na část klientskou a serverovou tak, aby pro většinu interakcí koncového uživatele aplikace nebylo nutné komunikovat se serverem; tím lze minimalizovat čas čekání uživatele na server.
Další podstatnou výhodou je, pokud aplikační architektura s automatickou údržbou logické integrity s konkurenčním zpracováním používá poptávkové řízení, že data jsou zasílány jen klientům, kteří je fakticky potřebují a stejně tak i informace o změnách. Tím jsou minimalizovány celkové komunikační náklady.
Výhodná je rovněž skutečnost, že o změnách v datech, iniciovaných jedním uživatelem, se s minimální prodlevou dozví ostatní uživatelé, kteří tato data poptávají.

Systém může být doplněn o mechanismus, který v případě přerušení spojení mezi klientem a serverem požadavky a informace o změně hodnoty ukládá do paměti v místě vzniku této zprávy; a automaticky je posílá při následném obnovení spojení. Je to výhodné, protože jednotliví klienti, stejně jako server, mohou být přechodně ukončeni nebo odpojeni od sítě, aniž by se narušila konzistence celého systému klientů a serverů. Navíc s klientem může uživatel, často bez výrazné ztráty funkčnosti, nadále pracovat, i když je od serveru odpojen. Většina uživatelových interakcí nevyžaduje komunikaci se serverem.

Výhodná je též možnost rozdělení aplikace na libovolný počet úrovní pracování; kromě popisovaných dvou úrovní klient – server lze aplikaci provozovat například na čtyřech úrovních, mezi klienta a server lze v takovém případě vložit na straně klienta vrstvu zástupce, který soustřeďuje komunikaci se serverem pro více klientských počítačů a na straně serveru vložit vyrovnávací servery, které rozdělují zpracování aplikační logiky na více vyrovnávacích počítačů. Dochází k rozložení výkonu na více počítačů na straně serveru; na klientské straně se jedná o minimalizaci komunikačních nákladů.
Řešení podle tohoto vynálezu je využitelné i při integraci aplikací od různých dodavatelů. Při integraci aplikací od různých dodavatelů je nejpracnější vývoj algoritmu pro automatické zapracování informací z jednoho systému do druhého systému v reálném čase. Tato pracnost je tak velká a roste takovým způsobem s velikostí aplikace, že integrace aplikací v reálném čase se realizuje jen zřídka a s velkými náklady.

Pro odstranění tohoto problému se s výhodou využije Aplikační architektura s asynchronním komunikačním protokolem a automatickou synchronizací a doplní se následovně:
V okamžiku deklarace klientského vztahu se určí, na základě standardizovaného způsobu identifikace (například URL), na který server má klientský vztah směřovat své požadavky. Tímto způsobem může aplikace A automaticky využívat data z aplikace B. V aplikaci A postačuje pouze doplnit zástupce pro potřebná data z aplikace B o klientský vztah s identifikací počítače, který zpracovává aplikaci B. V aplikaci B je třeba doplnit vztahy server k datům, které se dávají k dispozici k integraci s jinými aplikacemi. Serverové vztahy je možné doplnit o řízení přístupu, to je algoritmy, které rozhodnou, zda a za jakých podmínek a kterým klientům budou data poskytovat. Zpracování informací mezi integrovanými aplikacemi probíhá automaticky v reálném čase, s minimálními náklady na vývoj propojení.

Průmyslová využitelnost

Vynález je využitelný při udržování logické integrity mezi objekty v interaktivních aplikacích počítačů, včetně v systémech sdílení dat v sítích.

PATENTOVÉ NÁROKY

1. Způsob udržování logické integrity mezi objekty v interaktivních aplikacích počítačů, vyznačující se tím, že logické vztahy mezi jednotlivými objekty se pouze deklarují vazbami mezi jednotlivými objekty aplikace tak, že se vybírají vztahy ze souboru obecných, předem definovaných základních vztahů a jednotlivé objekty se připojují na výstupy nebo vstupy daného vztahu, načež každému základnímu vztahu se přiřadí obecný předem definovaný prostředek pro naplňování a udržování těchto vztahů při jejich používání, kde objekty informují tyto prostředky, přiřazené ke vztahům, které na jejich hodnotách závisejí, o změnách svých hodnot, a prostředky reagují na změny v hodnotách svých vstupů a za účelem zachování integrity vztahů mění své výstupy a pracují konkurenčně, kdy několik prostředků může být aktivních současně, takže při běhu aplikace se současně zpracovává vstup uživatele a provádí se udržování integrity vztahů.

2. Způsob podle nároku 1, vyznačující se tím, že zpracování prostředků pro naplňování a udržování logických vztahů, tvořených algoritmy, při konkurenčním zpracovávání těchto prostředků a vstupu uživatele se rozdělí na krátké časové úseky, mezi kterými může být algoritmus přerušen a během nichž přerušen být nemůže a okolní data se tedy nemění, a aplikace se řídí centrální smyčkou, určující okamžik, kdy probíhá zpracování vstupů od uživatele a kdy se provádějí kroky uvedených algoritmů a kterých algoritmů, přičemž jednotlivé akce se řadí časově za sebou.

3. Způsob podle nároku 1 nebo 2, vyznačující se tím, že při informování prostředků pro naplňování a udržování logických vztahů mezi jednotlivými objekty o změnách v okolí se pro výpočet nové hodnoty využijí před změnami vypočítané hodnoty.

4. Způsob podle některého z nároků 1 až 3, vyznačující se tím, že prostředky pro naplňování a udržování logických vztahů mezi jednotlivými objekty se řídí poptávkou po hodnotách objektu tak, že naplňování a udržování vztahu probíhá jen při poptávce po výsledku, a po ukončení poptávky se naplňování, případně udržování ukončí.

5. Způsob podle některého z nároků 1 až 4, vyznačující se tím, že všechny výstupy, mezivýsledky a stavové informace prostředků pro naplňování a udržování logických vztahů mezi jednotlivými objekty se ukládají do trvalé paměti.

6. Způsob podle některého z nároků 1 až 5, vyznačující se tím, že při konkurenčním zpracování prostředků pro udržování logických vztahů mezi jednotlivými objekty a vstupu uživatele se do systému zařadí garbage collector pro mazání neužitečných hodnot, kterým se na základě informací o volné paměti počítače a dalších informací iniciuje mazání neužitečných hodnot.

7. Použití způsobu podle některého z nároků 1 až 6 pro systém pro sdílení dat v sítích, zahrnující server k centrálnímu ukládání sdílených dat a nejméně jeden klient, na kterém je provozována aplikace, která využívá sdílená data a zpracovává informace o změnách sdílených dat, kdy klienty jsou alespoň přechodně spojeny sítí, opatřený pro aplikaci pro přístup na sdílená data z klientu vyrovnávací pamětí.

8. Systém pro sdílení dat v sítích, zahrnující server k centrálnímu ukládání sdílených dat a nejméně jeden klient, na kterém je provozována aplikace, která využívá sdílená data a zpracovává informace o změnách sdílených dat, kdy klienty jsou alespoň přechodně spojeny sítí, opatřený pro aplikaci pro přístup na sdílená data z klientu vyrovnávací pamětí podle nároku 7, vyznačující se tím, že pro použití způsobu údržby logické integrity v interaktivních aplikacích počítačů podle některého z nároků 1 až 6 zahrnuje dále soubor obecných, předem definovaných základních vztahů speciální klientský vztah s nejméně jedním výstupem a žádným vstupem, přičemž všechna sdílená data v klientské aplikaci jsou reprezentována objektem s připojeným klientským vztahem, pracujícím jako vyrovnávací paměť.

9. Systém pro sdílení dat v sítích, zahrnující server k centrálnímu ukládání sdílených dat a nejméně jeden klient, na kterém je provozována aplikace, která využívá sdílená data a zpracovává informace o změnách sdílených dat, kdy klienty jsou alespoň přechodně spojeny sítí, opatřený pro aplikaci pro přístup na sdílená data z klientu vyrovnávací pamětí podle nároku 7 nebo 8, vyznačující se tím, že pro

použití způsobu údržby logické integrity v interaktivních aplikacích počítačů podle některého z nároků 1 až 6 zahrnuje dále soubor obecných, předem definovaných základních vztahů speciální serverový vztah s nejméně jedním vstupem a žádným výstupem, přičemž ke všem objektům, jejichž hodnoty jsou sdíleny, jsou připojeny serverové vztahy pro reagování na požadavky klientů.

10. Způsob sdílení dat v systému podle nároku 7, vyznačující se tím, že v případě, kdy aplikace požaduje hodnotu, která ještě není uložena ve vyrovnávací paměti, zašle se po síti požadavek o tuto hodnotu na server a zároveň se aplikaci vrátí libovolná hodnota zpět, načež jakmile se ze serveru vrátí odpověď, pak pokud tato odpověď není shodná s hodnotou, která již dříve byla vrácena aplikaci, tak se aplikaci oznámí změna hodnoty a hodnota obdržená ze serveru se zapamatuje ve vyrovnávací paměti, přičemž jakmile klient obdrží po síti od serveru informace o změně hodnoty, oznámí se tato změna aplikaci a z vyrovnávací paměti se hodnota vymaže nebo se označí za neplatnou tak, aby při dalším požadavku aplikace na tuto hodnotu došlo k novému požadavku na server, a jakmile aplikace na klientu požaduje změnit některou sdílenou hodnotu, tak se nová hodnota okamžitě uloží do vyrovnávací paměti, přičemž aplikace je o změně hodnoty okamžitě informována a zároveň se na server zašle požadavek na změnu této hodnoty, a dokud klient od serveru neobdrží potvrzení, že server jeho požadavek zpracoval, ignorují se veškeré zprávy ze serveru o změně dané hodnoty, s tím, že požadavky klientů na zaslání hodnot se přijímají po síti serverem, který hodnoty zasílá zpět, jakmile je sám zjistí a zároveň si pamatuje, kterým klientům které hodnoty odeslal, a jakmile se změní některé hodnoty na serveru, odesílá se tato informace ze serveru těm klientům, kterým se odeslala původní hodnota a pokud se těmito klienty původní hodnota znovu nevyžádá, potom další informace o změnách dané hodnoty od serveru se jim nadále nezasílají, přičemž server přijímá od klientů požadavky na změnu své hodnoty, které se zpracovávají tak, že server danou hodnotu změní a poté odešle klientovi, od kterého požadavek obdržel, informaci o provedené změně a zároveň se tato informace o změně hodnoty odesílá i dalším klientům, kterým server dříve odeslal původní hodnotu.

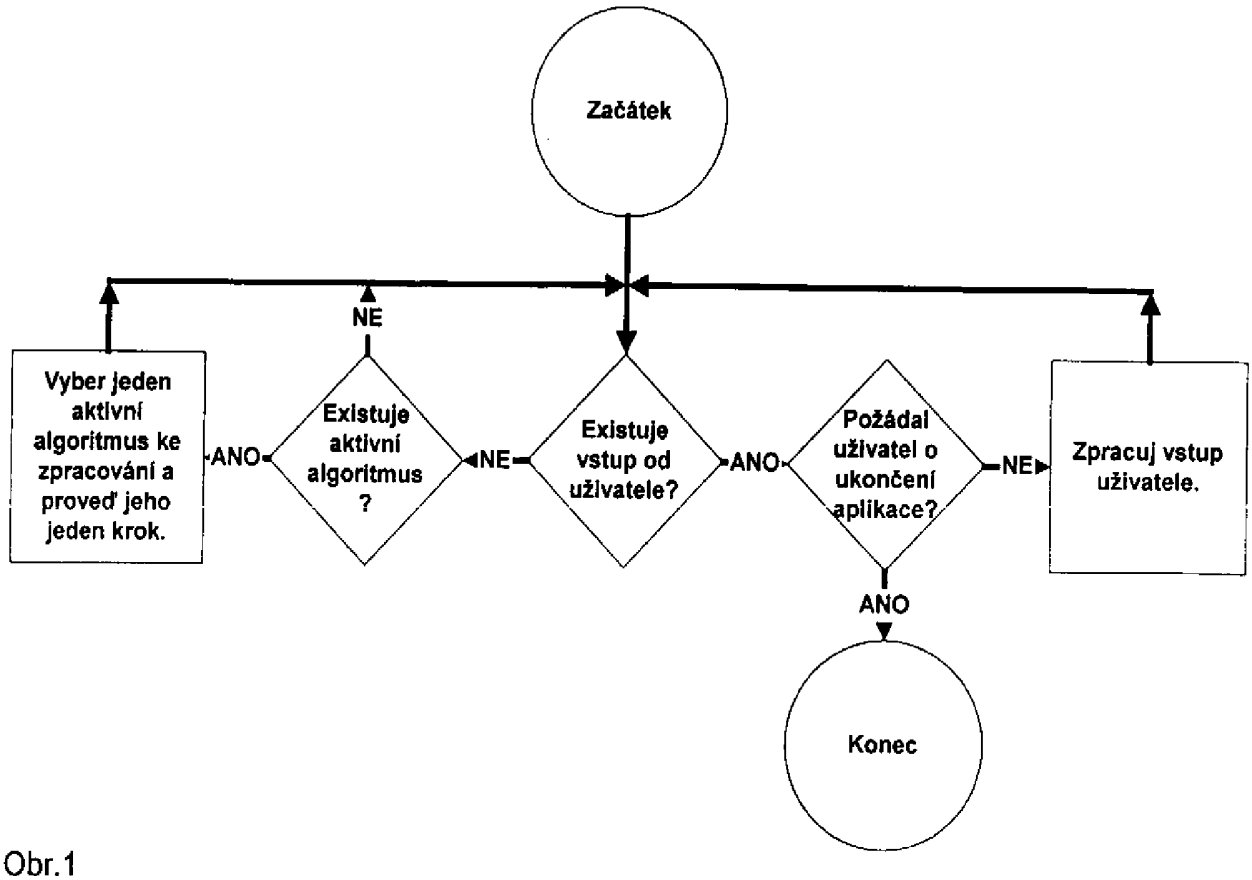
11. Způsob sdílení dat v systému podle některého z nároků 7 až 10, vyznačující se tím, že při přerušení spojení mezi klientem a serverem se požadavky a informace

o změně hodnoty ukládají do paměti v místě vzniku této zprávy a odesílají se při následném obnovení spojení.

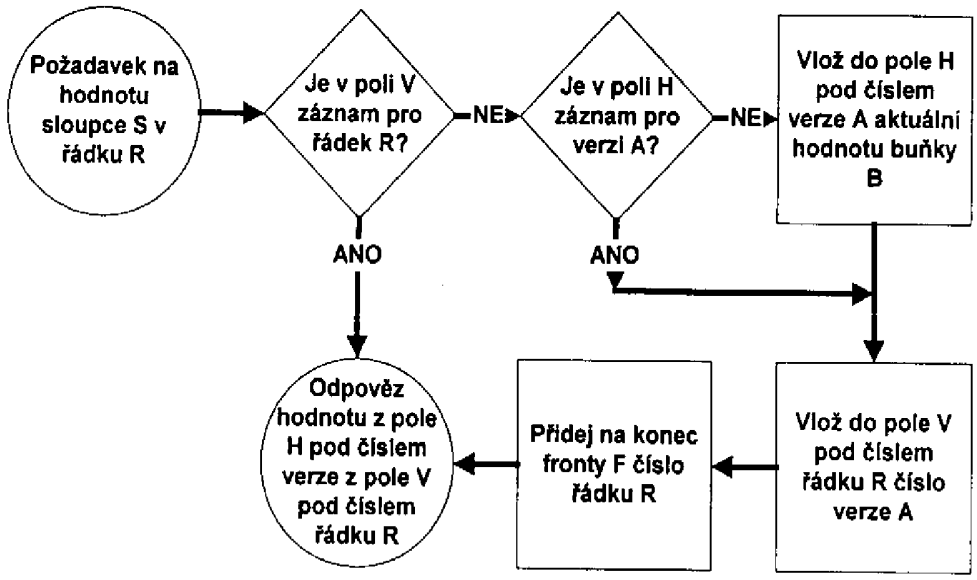
12. Způsob sdílení dat v systému podle některého z nároků 7 až 11, vyznačující se tím, že aplikace se rozdělí na větší počet úrovní než klient a server.

13. Použití způsobu sdílení dat podle některého z nároků 7 až 12 pro aplikace, sdílející data z více než jednoho serveru.

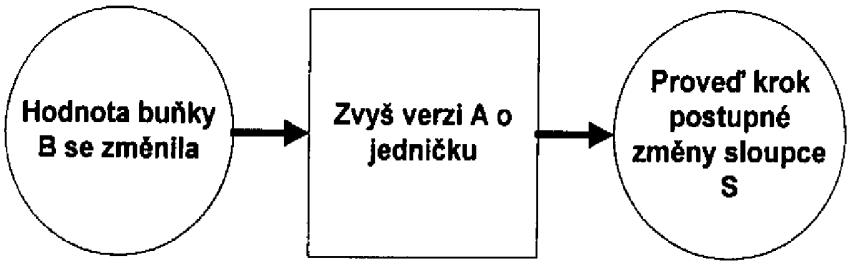
14. Použití způsobu sdílení dat podle nároku 13, kdy, po doplnění zástupce pro potřebná data z jiné aplikace o klientský vztah s identifikací počítače, zpracovávajícího tuto aplikaci, a doplnění v ní o vztahy server k datům, které se dávají k dispozici k integraci s jinými aplikacemi, se v okamžiku deklarace klientského vztahu určí, na který server má klientský vztah směřovat své požadavky.


Začátek
Vyber jeden aktivní algoritmus ke zpracování a proveď jeho jeden krok.
ANO
Existuje aktivní algoritmus ?
NE
NE
Existuje vstup od uživatele?
ANO
Požádal uživatel o ukončení aplikace?
NE
Zpracuj vstup uživatele.
ANO
Konec

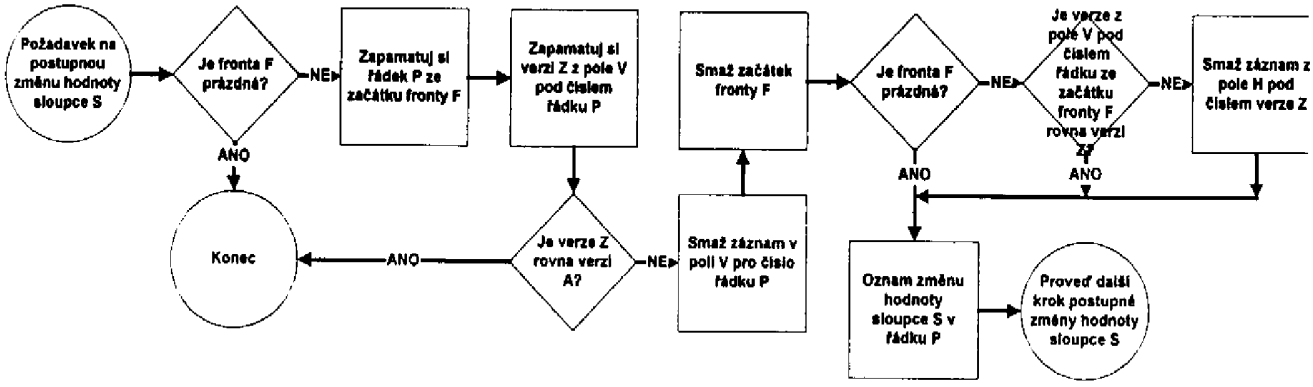
Obr.1


Požadavek na hodnotu sloupce S v řádku R
Je v poli V záznam pro řádek R?
NE
Je v poli H záznam pro verzi A?
NE
Vlož do pole H pod číslem verze A aktuální hodnotu buňky B
ANO
ANO
Odpověz hodnotu z pole H pod číslem verze z pole V pod číslem řádku R
Přidej na konec fronty F číslo řádku R
Vlož do pole V pod číslem řádku R číslo verze A

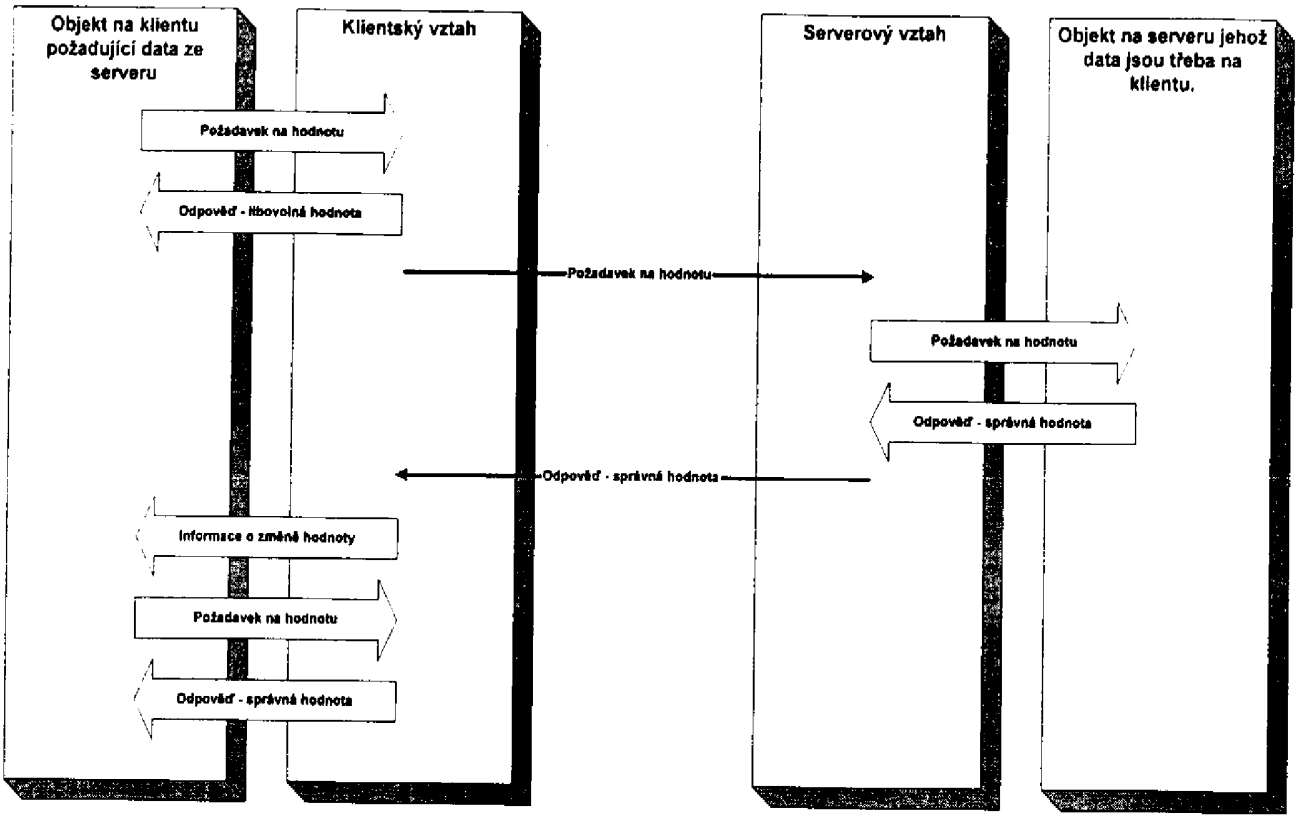
Obr.2


Hodnota buňky B se změnila
Zvyš verzi A o jedničku
Proveď krok postupné změny sloupce S

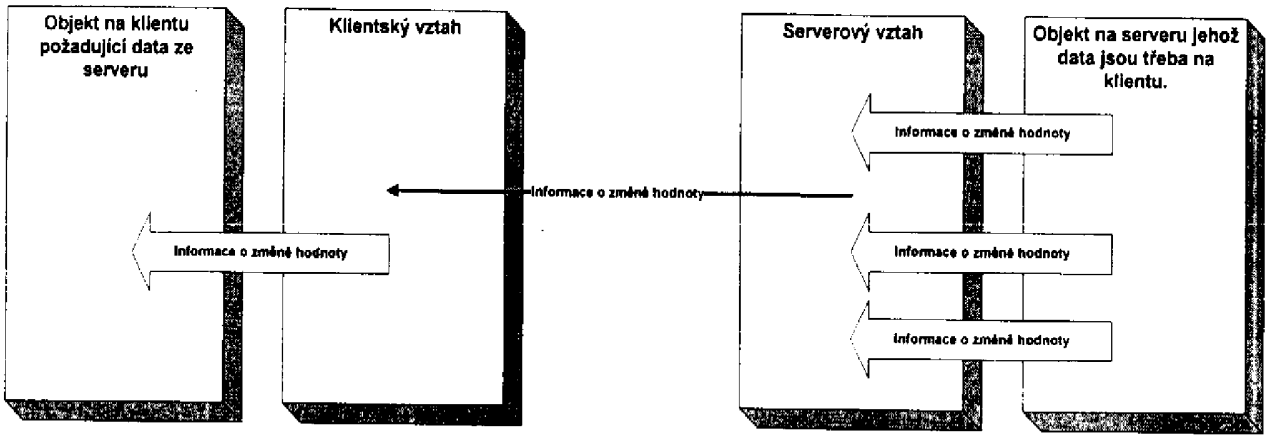
Obr.3


Požadavek na postupnou změnu hodnoty sloupce S
Je fronta F prázdná?
NE
ANO
Zapamatuj si řádek P ze začátku fronty F
Zapamatuj si verzi Z z pole V pod číslem řádku P
Smaž začátek fronty F
Je fronta F prázdná?
NE
ANO
Je verze z pole V pod číslem řádku ze začátku fronty F rovna verzi Z?
NE
ANO
Smaž záznam z pole H pod číslem verze Z
Konec
ANO
Je verze Z rovna verzi A?
NE
Smaž záznam v poli V pro číslo řádku P
Oznam změnu hodnoty sloupce S v řádku P
Proveď další krok postupné změny hodnoty sloupce S

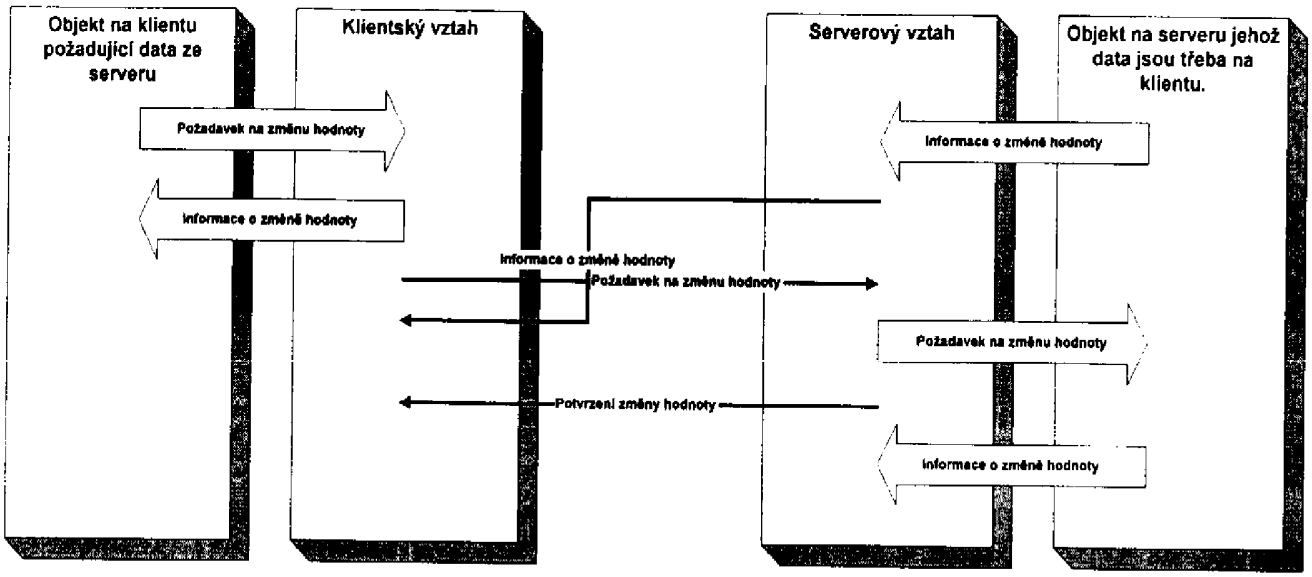
Obr.4


Objekt na klientu požadující data ze serveru
Klientský vztah
Serverový vztah
Objekt na serveru jehož data jsou třeba na klientu.
Požadavek na hodnotu
Odpověď - libovolná hodnota
Požadavek na hodnotu
Požadavek na hodnotu
Odpověď - správná hodnota
Odpověď - správná hodnota
Informace o změně hodnoty
Požadavek na hodnotu
Odpověď - správná hodnota

Obr.5.


Objekt na klientu požadující data ze serveru
Klientský vztah
Serverový vztah
Objekt na serveru jehož data jsou třeba na klientu.
Informace o změně hodnoty
Informace o změně hodnoty
Informace o změně hodnoty
Informace o změně hodnoty
Informace o změně hodnoty

Obr.6


Objekt na klientu požadující data ze serveru
Klientský vztah
Serverový vztah
Objekt na serveru jehož data jsou třeba na klientu.
Požadavek na změnu hodnoty
Informace o změně hodnoty
Informace o změně hodnoty
Požadavek na změnu hodnoty
Informace o změně hodnoty
Požadavek na změnu hodnoty
Potvrzení změny hodnoty
Informace o změně hodnoty

Obr.7