# AUTOMATIZOVANÁ TVORBA DATABÁZOVÝCH APLIKACÍ V PRAXI

**Miroslav Padalík**

Aura, s.r.o., Úvoz 56, 602 00 Brno, ČR

**Abstrakt**

Příspěvek pojednává o automatizovaném generování vícevrstvých databázových aplikací s grafickým uživatelským rozhraním pomocí CASE nástrojů. Poukazuje na to, že technologie založená na generování aplikací představuje ve spojení s komponentovým vývojem jeden z nejefektivnějších způsobů tvorby informačních systémů.

V několika samostatných kapitolách je popsáno generování prezentační, aplikační a databázové vrstvy aplikace, přičemž je zdůrazněna jejich plná integrace. Pro demonstraci jsou uvedeny diagramy v CASE nástroji spolu s formulářem, který z nich byl vygenerován.

Na závěr jsou pro lepší představu o použité technologii uvedeny obrázky zachycující reálné vývojové prostředí a diagram znázorňující schéma tvorby informačních systémů od úvodního zadání po vytvoření spustitelných aplikací.

## 1. Úvod

Stále větší složitost vytvářených databázových systémů vede k rostoucí potřebě důkladné analýzy. U větších systémů se většinou pro analýzu používají podpůrné CASE nástroje. Současné CASE nástroje sice umožňují provedení analýzy pomocí různých diagramů, samotné programování však musí být většinou provedeno ručně. Některé CASE nástroje obsahují různé integrované generátory, většinou však jde jen o možnost vygenerování kostry aplikace (hlavičky objektů nebo funkcí), přičemž samotné programování zůstává na vývojových pracovnících.

Tlak na rychlost vývoje a snižování nákladů vede ke snaze použít takový nástroj, který by umožňoval rychlou a efektivní tvorbu kvalitních aplikací s co nejnižšími náklady na údržbu. Nástroj by měl maximálně využívat výsledky analýzy a být přitom snadno zvládnutelný v běžném vývojovém týmu. Většina dostupných nástrojů kladené požadavky nesplňuje a proto nezbývá, než si existující nástroje přizpůsobit nebo si vytvořit vlastní. To je však velmi náročné jak časově, tak finančně a většina firem proto přistupuje na různé kompromisy.

Naše firma se již několik let zabývá tvorbou informačních systémů vytvořených na klíč dle konkrétních požadavků zákazníka. Snaha po co nejefektivnějším vývoji aplikací vedla postupně k vybudování integrované technologické linky pokrývající celý životní cyklus tvorby aplikací a umožňující maximálně automatizovaný vývoj od fáze získávání požadavků, přes analýzu, návrh a programování až po testování, nasazení a údržbu aplikací. Součástí této technologické linky se stal námi vyvinutý generátor vícevrstvých aplikací s grafickým uživatelským rozhraním VIAGEN, který maximálně využívá výsledků analýzy provedené v CASE nástroji a dokáže z příslušných diagramů vygenerovat všechny vrstvy aplikace, z nichž se okamžitě stává plně funkční aplikace. Spolu s tímto generátorem byla vytvořena sada komponent umožňující zapouzdření funkčnosti nezávislé na aplikaci modelované v CASE nástroji.

Vícevrstvé databázové aplikace obsahují nejméně tři základní vrstvy – databázovou, aplikační a prezentační. Každá z nich má svá specifika, ke kterým je třeba při vývoji přihlížet.

Následující kapitoly pojednávají o efektivním způsobu automatizovaného vývoje jednotlivých vrstev aplikací a jeho praktickém použití v naší firmě. Kromě databázové, aplikační a prezentační vrstvy stojí za zmínku také komunikační vrstva. Ta je však řešena na úrovni obecného aplikačního serveru a obecných komunikačních komponent a knihoven a je z hlediska vývojářů transparentní.

## 2. Tvorba databázové vrstvy

Pro realizaci databázové vrstvy je k dispozici mnoho databázových serverů, které nabízí širokou škálu možností v oblasti práce s daty. Pokud jsou používané databáze dobře navržené, je možné bez problémů použít jejich data v aplikační vrstvě, která je zpracuje a v potřebném tvaru přenese na prezentační vrstvu, kde jsou uživateli k dispozici.

Před vlastním použitím je nejdříve potřeba databázi navrhnout, vytvořit a naplnit daty. Kvalitní návrh databáze je velmi důležitý, neboť může značně ovlivnit odezvy a možnosti cílové aplikace. Správný postup vedoucí k dobrému výsledku se skládá z několika kroků. Nejprve je v CASE nástroji na úrovni diagramů vytvořen logický datový model, který vyplývá z analýzy. Ten je dále transformován na implementační datový model, v němž už jsou zohledněna implementační hlediska. Z implementačního modelu jsou pak automaticky vygenerovány skripty pro vytvoření databáze.

V dnešní době dokáže modelovat datový model většina CASE nástrojů, a to včetně možnosti generovat z něj skripty pro vytváření databáze. Problémy bývají pouze s velikostí datového modelu a s rozsahem podpůrné funkčnosti. Proto je velmi důležité, aby byl použitý CASE nástroj dostatečně otevřený a bylo možno jej přizpůsobit konkrétním podmínkám.

Otevřenosti CASE nástroje jsme využili i my a standardně dodávaný generátor databází jsme rozšířili jak kapacitně, tak funkčně. Nyní je běžně používán pro tvorbu velmi rozsáhlých databází složených z řádově tisíců entit a desetitisíců atributů. Pro vygenerování skriptů se nabízí celá řada různých variant závisejících na lokalitě, kde bude databáze umístěná, na typu aplikace, která s ní pracuje apod. Navíc jsou automaticky generovány parametry pro optimální nastavení velikosti datových ploch pro tabulky podle předpokládaného počtu jejich záznamů. Samozřejmostí je podpora všech typů vazeb včetně vazeb s kardinalitou m:n, generování vložených procedur a triggerů pro ošetření nastavených politik vazeb (rekurze a restrikce) a podpora replikací dat (online i offline) mezi vzdálenými servery. Vygenerované skripty obsahují u každé tabulky a sloupce přehledné informace o odpovídajících entitách, atributech a vazbách použitých v diagramech s datovým modelem.

## 3. Tvorba prezentační vrstvy

Tvorba prezentační vrstvy bývá často softwarovými firmami podceňována, na což pak doplatí koncový uživatel. Naše technologie umožňuje modelovat prezentační vrstvu v CASE nástroji a následně z něj generovat přehledné uživatelsky příjemné formuláře a tiskové sestavy.

### *3.1 Nástroje pro interaktivní tvorbu formulářů*

Pro tvorbu prezentační vrstvy existuje řada nástrojů, které umožňují interaktivní tvorbu uživatelských formulářů. Většina těchto nástrojů nabízí vývojářům možnost rychlé tvorby

formulářů pouhým sestavením grafických objektů pomocí myši. Tato metoda se může na první pohled jevit jako velmi efektivní a může na reklamních prezentacích učinit na nezasvěcené diváky velký dojem. Problém je ovšem v tom, že při předvádění nástrojů jsou již většinou prezentované aplikace buď předpřipravené, nebo jsou natolik jednoduché, že s reálnými systémy nemají mnoho společného. Když pak přijde na tvorbu reálné aplikace, nestačí se vývojový pracovník divit, jak se začíná vývoj komplikovat, co všechno vychvalovaný nástroj neumí a kolik ruční práce bude třeba do vývoje investovat.

Jedním ze základních nedostatků tvorby aplikací pomocí běžných vývojových nástrojů je zastínění potřeby důkladné analýzy obsahu jednotlivých formulářů. Odhadnout přibližný obsah a vzhled formuláře přímo při jeho tvorbě není většinou složité, ale vytvořit konečnou podobu formuláře tak, aby byly splněny všechny požadavky na funkčnost a snadnou ovladatelnost aplikace již snadné není. Vývojáři se musí kromě samotného datového obsahu formuláře zabývat jeho vzhledem, vzájemným umístěním jednotlivých datových elementů a způsobem jejich grafické prezentace. Po vytvoření takového formuláře se často zjistí, že se na některé datové elementy pozapomnělo, a že není jasná vazba mezi použitými objekty. Cílový formulář bývá z hlediska datového obsahu nepřehledný a není úplně prokazatelné, že pokryl původní požadavky. Rovněž dochází k častým a nemalým změnám, které výsledek činí ještě méně přehledným.

### *3.2 Návrh prezentační vrstvy pomocí CASE nástroje*

U lepších vývojových nástrojů bývá zvykem alespoň v dokumentaci zdůraznit, že důkladná analýza a návrh všech formulářů je základním klíčem k úspěchu. To však nestačí a je třeba tuto ideu softwarově podpořit. K tomuto účelu mohou posloužit takové CASE nástroje, které umožňují modelování uživatelského rozhraní. Nejvhodnější formu pro takové modelování tvoří grafické diagramy s přímou vazbu na logický datový model aplikace, umožňující zachytit obsah formulářů ve formě nezávislé na cílové platformě.

Pokud už se vývojový tým rozhodne použít při návrhu aplikace CASE nástroje, je to většinou bez generátoru formulářů a tým programátorů musí namodelované formuláře celé ručně vytvořit. V lepším případě je k dispozici jednoduchý generátor, který umožní vygenerovat triviální podobu formulářů, přičemž většina práce stejně zůstane na ruční tvorbě.

### *3.3 Tvorba prezentační vrstvy pomocí generátoru*

Před zahájením vývoje vícevrstvých aplikací s grafickým uživatelským rozhraním provedla naše firma průzkum trhu s vývojovými nástroji a vybrané kandidáty vyzkoušela na menších aplikacích. Jelikož žádný s dosažitelných nástrojů zdaleka nepokryl veškeré požadavky kladené na vývoj rozsáhlých databázových aplikací, a to zejména v komplexnosti a efektivnosti vývoje, rozhodli jsme se pro vytvoření vlastního nástroje, který by maximálně automatizoval celou fázi tvorby aplikací od logického návrhu v CASE nástroji až po vytvoření hotové spustitelné aplikace. Na základě tohoto rozhodnutí byl vytvořen výkonný generátor VIAGEN, který umožňuje ve spojení se sadou připravených komponent a knihoven automatizovanou tvorbu libovolně velkých vícevrstvých aplikací s grafickým uživatelským rozhraním.

Vstupem pro tvorbu prezentační vrstvy pomocí tohoto generátoru jsou logický model databáze a model uživatelského rozhraní vytvořené v CASE nástroji. Výstupem je objektově orientovaný zdrojový kód definující veškeré vizuální i nevizuální prvky formuláře včetně metod pro jejich obsluhu. Součástí kódu jsou rovněž veškerá rozhraní mezi vrstvami aplikace.

Vygenerovanou aplikaci je možné ihned po kompilaci spustit a provozovat. V případě datových mřížek je možno přímo v grafickém vývojovém prostředí prohlížet data z aplikační vrstvy ještě před samotnou kompilací, aniž by tato data musela odpovídat databázovým tabulkám.

Do generátoru jsou zahrnuty sofistikované algoritmy umožňující optimální rozvržení jednotlivých datových bloků na generovaném formuláři. Přesto má návrhář možnost podle vlastního estetického cítění vzhled formulářů upravit, a to buď na úrovni CASE nástroje nebo přímo v grafickém pohledu na formulář metodou drag&drop. Pokud nastane potřeba model uživatelského rozhraní pozměnit přímo v diagramu nebo dojde k obecné změně vzhledu formulářů, má návrhář možnost formuláře přegenerovat s volbou zachování nebo zrušení změn provedených ručně v grafickém pohledu na formulář.

### *3.4 Základní vlastnosti vygenerovaných formulářů*

Uživatelské formuláře mohou obsahovat několik lišt obsahujících menu, ikony a různé typy informací. Datová část formuláře může být rozdělena do libovolného počtu navzájem datově provázaných bloků. Jejich vzájemná vazba většinou odpovídá vztahům v databázi, může však být vytvořena zcela obecně a nezávisle na databázi.

Datové bloky mohou být pro přehlednost rozmístěny do jednotlivých záložek, přičemž každý blok může být na více záložkách v různých podobách. Pohyb po jeho záznamech je pak na všech záložkách automaticky synchronizován.

Každý datový blok může obsahovat jeden záznam reprezentovaný různě rozmístěnými datovými objekty nebo více záznamů reprezentovaných datovou mřížkou nebo kombinací mřížky a jiných objektů. Každý záznam může obsahovat libovolné množství polí z řídící entity a z libovolného množství číselníkových polí. Číselníková pole mohou být svázána přes několik vazeb nebo přes vazby neobsažené v databázi. Do číselníkových polí je možno vkládat hodnoty buď jednoduchou formou pomocí výběru z nabízených hodnot, nebo pomocí číselníkových formulářů, které poskytují širokou škálu funkčností. V číselníkových formulářích je možno plnohodnotně filtrovat a třídit data a vyvolávat z nich vnořené číselníkové formuláře do libovolné hloubky.

Nad jednotlivými datovými bloky nebo nad skupinou bloků lze provádět dotazy pomocí dotazovacích formulářů. Na základě uživatelem zadaných omezujících podmínek jsou vybrána požadovaná data, nad kterými lze provádět všechny dostupné operace. Z dotazovacích formulářů lze rovněž vyvolávat číselníkové formuláře pro výběr omezujících hodnot.

Každý blok může být navázán na množinu aplikačních akcí, prostřednictvím kterých je dynamicky řízen přístup uživatele k datům a operacím. Položky menu, ikony a další řídící objekty odpovídající akcím, které nemá uživatel momentálně povoleny, jsou deaktivovány nebo zcela skryty. Dostupnost objektů na formuláři se může při běhu aplikace dle stanovených pravidel měnit.

Datová pole mohou být různého typu. Generátor rozlišuje pole určená pouze pro čtení, modifikovatelná pole a pole, která musí být povinně vyplněna. Jednotlivé typy polí jsou od sebe barevně odlišeny. Každé pole může dynamicky měnit svůj typ v závislosti na okolních podmínkách.

Formuláře je možno libovolně propojovat mezi různými subsystémy nebo projekty, a není je proto nutno vytvářet vícekrát.

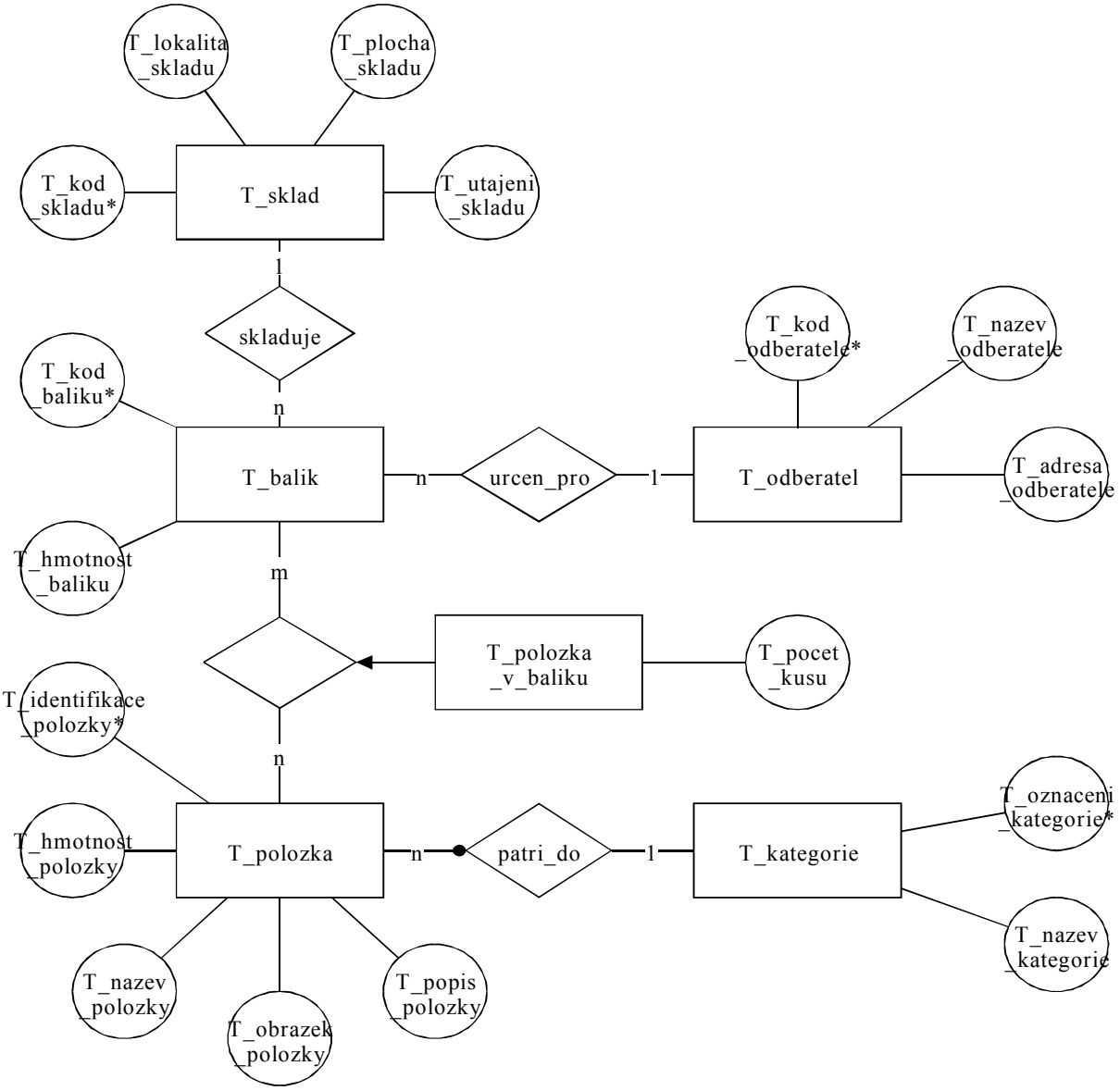
### *3.5 Příklad aplikace vytvořené pomocí generátoru*

Pro účely tohoto příspěvku byla v CASE nástroji namodelována jednoduchá aplikace, na které je možno demonstrovat vstupy a výstupy generátoru VIAGEN. Tato aplikace byla natolik zjednodušena, aby mohla ve snadno pochopitelné formě demonstrovat nejzákladnější principy použití generátoru.

Na obrázku Obr. 1 je zachycen zjednodušený logický datový model skladu ve formě ER-diagramu, přičemž:

- obdélníky značí entity, ze kterých budou vygenerovány databázové tabulky,
- kolečka značí atributy entit, z nichž budou vygenerovány sloupce tabulek
- kosočtverce značí vazby mezi entitami, které budou mít vliv na dovážené sloupce a klíče.

Názvy entit a atributů jsou uvedeny s předponou T_, která aplikaci identifikuje v rámci skupiny ostatních aplikací pracujících se stejnou databází. Model není pro jednoduchost normalizován.




Obr. 1: Zjednodušený logický datový model skladu

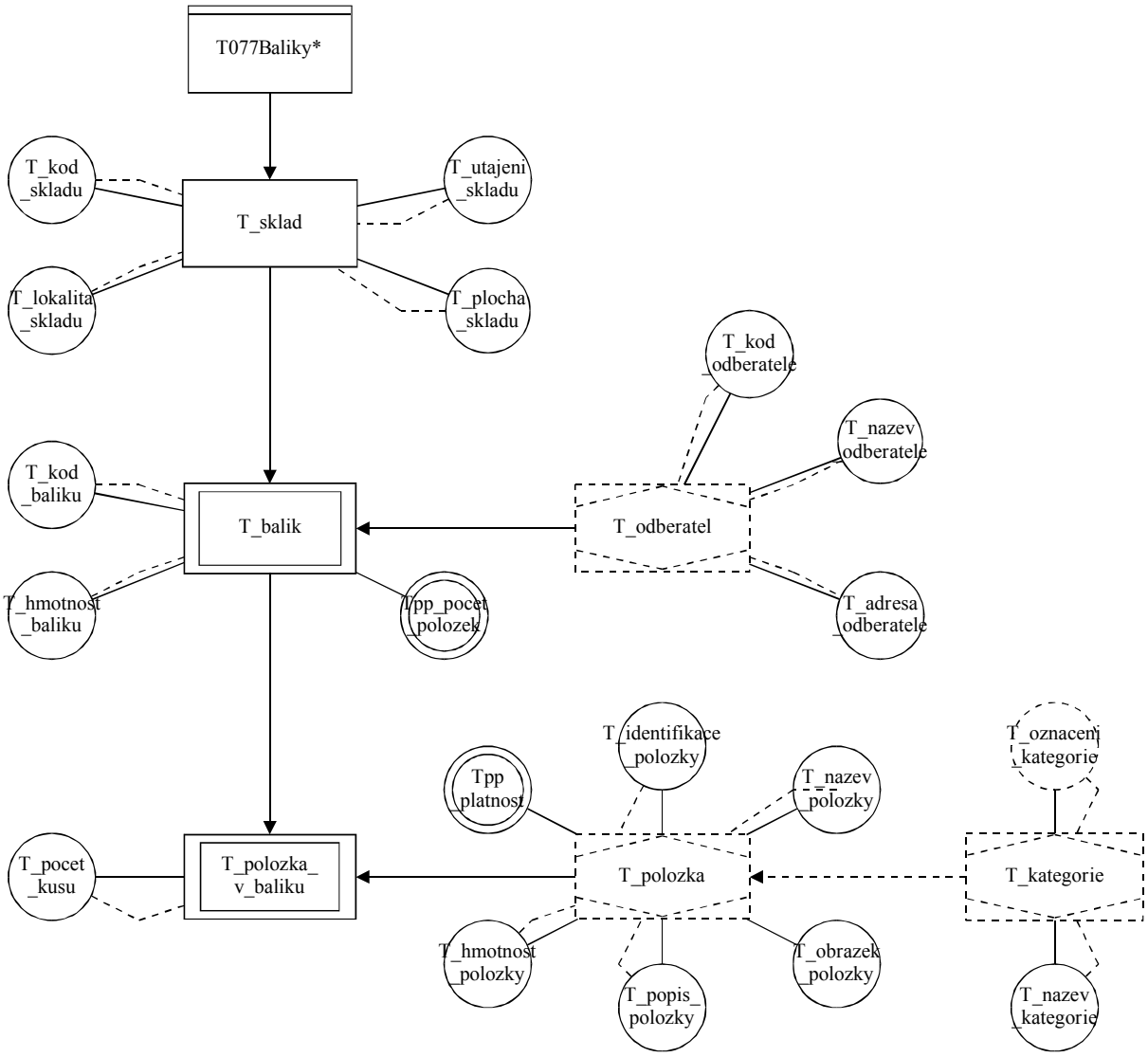
Na uvedeném obrázku je zachycen sklad, ve kterém se skladují různé balíky, přičemž balík může představovat např. paletu se zbožím. Z obrázku lze získat následující informace:

Na uvedeném obrázku je zachycen sklad, ve kterém se skladují různé balíky, přičemž balík může představovat např. paletu se zbožím. Z obrázku lze získat následující informace:

- jeden sklad může skladovat jeden nebo více balíků,
- každý balík může mít svého odběratele, přičemž pro stejného odběratele může být určeno více balíků,
- v každém balíku může být zabaleno několik položek a každá položka může být zabalena do více balíků
- každá položka patří do nějaké kategorie a více položek může patřit do stejné kategorie.

Našim záměrem je vytvořit formulář pro práci s balíky. Nejprve chceme namodelovat, co podstatného bude formulář obsahovat, aniž bychom se zdržovali tím, jak bude cílově vypadat. Na základě předchozí analýzy vytvoříme diagram s modelem uživatelského rozhraní pro požadovaný formulář (viz Obr. 2), kde:

- jednoduché obdélníky značí jednozáznamové bloky dat,
- dvojité obdélníky značí vícezáznamové bloky dat,
- jednoduchá kolečka značí datová políčka bloku,
- dvojitá kolečka značí počítaná pole,
- obdélníky s šestiúhelníkem značí číselníkové entity, z nichž budou dotahovány záznamy do odpovídajících polí bloku,
- přerušované spojnice označují pole, nad kterými bude moci uživatel provádět dotaz.

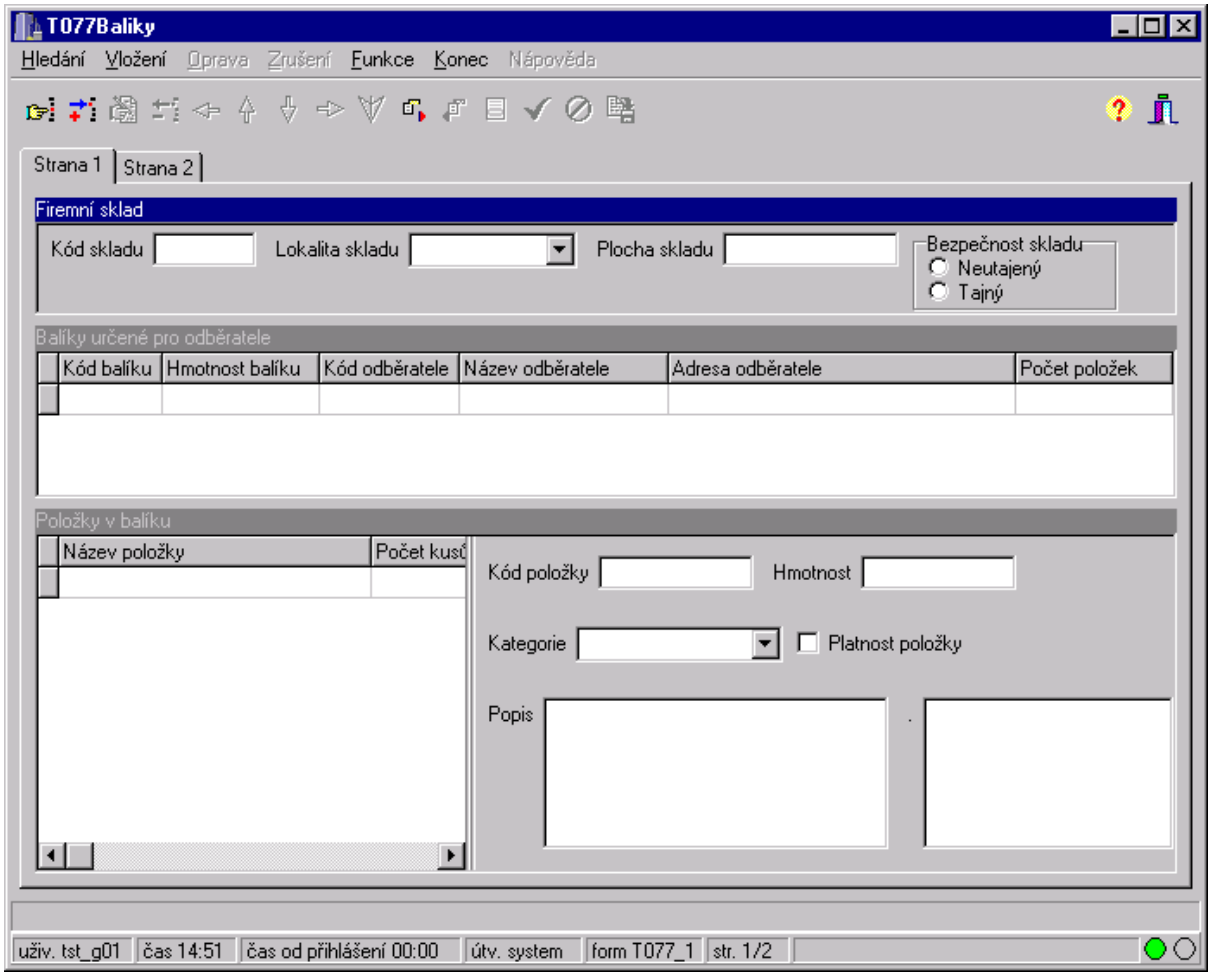



Obr. 2: Model uživatelského rozhraní pro formulář skladu

Uvedený diagram vyjadřuje skutečnost, že na formuláři bude v daný okamžik zobrazen jeden sklad, pro tento sklad budou zobrazeny balíky v něm skladované, přičemž u každého balíku bude uveden příslušný odběratel. Pro aktuálně vybraný balík budou na formuláři zobrazeny položky obsažené v balíku včetně jejich popisu, obrázku a kategorie, do které patří.

Jakmile je diagram s logickým obsahem formuláře hotov, je možné spustit generátor, který vygeneruje všechny vrstvy aplikace, po jejichž kompilaci vznikne plně funkční aplikace. Příklad formuláře odpovídajícího uvedeným diagramům je na obrázku Obr. 3. Tento formulář vznikl kompilací čistě vygenerovaného kódu bez jakýchkoliv ručních úprav.

Zajímavé je, že při tvorbě aplikace nebylo třeba napsat ani řádek kódu, a ani se zabývat skládáním vizuálních objektů do formuláře, ani rozhraním jednotlivých vrstev, ani aplikačním serverem a ani komunikací mezi jednotlivými počítači nebo servery.



Obr. 3: Ukázka vygenerovaného formuláře

## 4. Automatizovaná tvorba aplikační vrstvy

Aplikační vrstva může být fyzicky rozdělena do několika vrstev. Z hlediska obecného pohledu ji můžeme rozdělit na lokální aplikační logiku provozovanou na koncových počítačích a na serverovou aplikační logiku provozovanou na aplikačních serverech.

### *4.1 Lokální aplikační logika*

Lokální aplikační logika je tvořena speciální vrstvou umístěnou na klientském počítači, která je přímo navázaná na prezentační vrstvu. Do této vrstvy spadá především základní obsluha vizuálních objektů a funkčnost těsně svázaná s prezentačními objekty.

V případě jednodušších aplikací, u kterých není brán zřetel na komfort uživatelského rozhraní, může být lokální aplikační logika vytvořena interaktivně např. pomocí navigačních pomocníků. U větších aplikací je třeba tuto vrstvu naprogramovat, a to nejlépe s využitím předem vytvořených komponent, což však nezaručuje jednotné ovládání formulářů a vyžaduje to hodně času a námahy, nemluvě o množství chyb, kterých se mohou programátoři při ručním kódování dopustit.

Popisovaná technologie umožňuje vygenerovat lokální aplikační logiku automaticky, přičemž je plně využito všech vytvořených komponent. Při tvorbě generátoru byl kladen velký důraz na co nejúspornější kód a na to, aby opakující se funkčnosti byly v maximální

míře přesunuty do obecných knihoven a komponent. Tím se eliminuje hlavní nevýhoda většiny generátorů spočívající ve vytvoření mohutného a přitom nepřehledného kódu. Vývojový pracovník má možnost rozšířit generovanou funkčnost o vlastní kód a tím jednoduše ovlivnit chování prezentační vrstvy vůči uživateli.

### *4.2 Serverová aplikační logika*

Nejméně podporovanou vrstvou ze strany vývojových nástrojů byla donedávna serverová aplikační logika, do které spadá tzv. business logika. V poslední době se sice začínají objevovat různé aplikační servery, které by měly podporu vývoje v této oblasti zlepšit, jejich zavádění je však nákladné a klade vysoké nároky na přítomnost specialistů pro tuto oblast.

Významným pomocníkem při vytváření serverové aplikační logiky je v našem případě generátor, který dokáže zcela automaticky vygenerovat aplikační logiku pro veškeré standardní databázové operace a umožňuje vývojovým pracovníkům snadné začlenění rozšířené business logiky. Samozřejmostí je plná podpora toků dat mezi databázovou, aplikační a prezentační vrstvou, přičemž datové záznamy odcházející na prezentační vrstvu nemusí mít ekvivalent v databázové vrstvě.

Generátor plně podporuje komunikaci mezi jednotlivými vrstvami aplikace, i když tyto vrstvy leží na různých počítačích nebo serverech. Zároveň podporuje bezpečnost aplikace, distribuci dat a funkcí mezi servery a řadu dalších funkčností běžných v rozsáhlých informačních systémech.

### *4.3 Techniky generování aplikační logiky*

Výstupem generátoru je komplexní kód zahrnující veškerou základní aplikační logiku. Nejčastější technikou používanou programátory při práci s tímto nástrojem je doplňování programového kódu (např. obsluh událostí) do předem určených míst za účelem rozšíření základní funkčnosti aplikace. K tomuto účelu slouží předdefinované vstupní body, s jejichž pomocí může programátor vytvářet programové fragmenty s definovaným rozhraním, aniž by se musel zabývat tím, jak jej začlenit do rozsáhlého kódu aplikace. V rámci vstupních bodů má programátor k dispozici všechny potřebné údaje podle charakteru těchto bodů.

Technikou vstupních bodů se řeší zejména programátorská obsluha různých typů událostí na prezentační a aplikační vrstvě. Generátor obsah vstupního bodu automaticky přiřadí odpovídající události objektu definovaného v komponentě, nebo ho začlení na definované místo v generované metodě nebo funkci. Vstupním bodem se dá např. řešit obsluha události opuštění pole na formuláři, kde chce programátor zařadit kontrolu obsahu tohoto pole. Dalším typickým příkladem využití vstupních bodů je obsluha událostí na aplikačním serveru v různých okamžicích manipulace s daty – např. v rámci transakce těsně před modifikací záznamu, přičemž programátor má k dispozici jak tvar záznamu po načtení z databáze, tak nový tvar upravený uživatelem a rovněž záznamy z datově nadřízených bloků dat.

Další užitečnou technikou je náhrada celých funkcí a metod. Občas dojde k situaci, kdy je třeba standardně vygenerované funkce nebo metody (příp. funkce a metody z knihoven a komponent) pozměnit a přizpůsobit specifickým potřebám. Náhradu je možno provést přímo na úrovni CASE nástroje, přičemž ke konstrukci nové podoby funkce nebo metody je možné využít menší „stavební kostky“ zajišťující přesně definovanou funkčnost na elementárnější

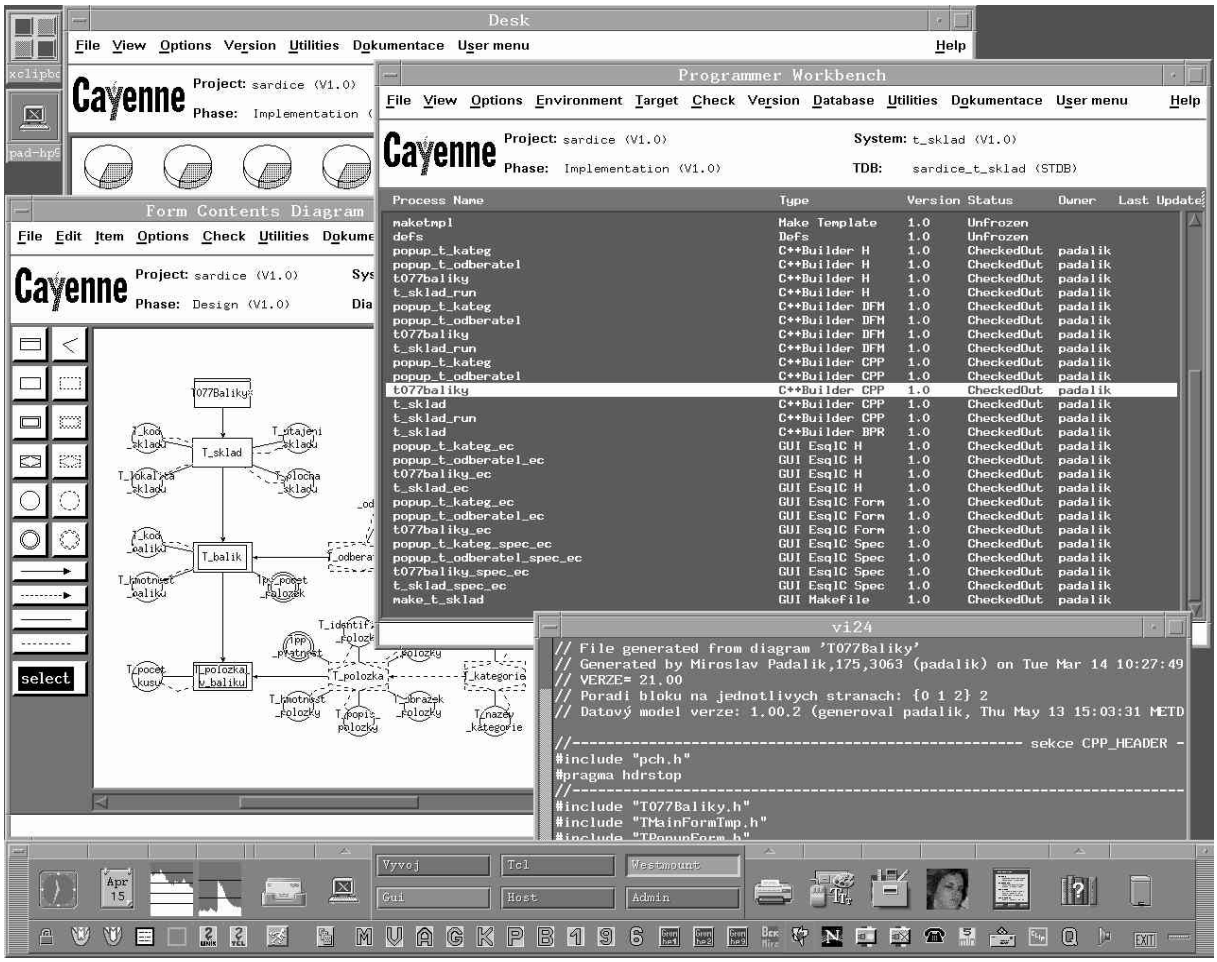
úrovni. Nová funkce nebo metoda pak většinou vznikne pouhým přeskupením těchto „stavebních kostek“.

Pokud je třeba jistou část vygenerovaného programového kódu pozměnit a přitom vývojáři nevyhovuje žádná z uvedených technik (stává se to zřídka, ale generátor kódu s tím musí počítat), je možné využít techniky substitucí, tedy přímo v diagramech nadefinovat automatickou náhradu zadaného kódu za jiný kód, a to buď jednorázovou nebo opakovanou. Při generování se pak všechny takto definované náhrady provedou a vývojář není nucen opakovaně měnit přímo vygenerovaný kód.

Poslední technikou je možnost přímých úprav vygenerovaných modulů buď v CASE nástroji nebo v prostředí kompilátoru, přičemž generátor je schopen při novém generování na základě rozboru cílového kódu rozeznat ruční úpravy kódu a podle charakteru změn oproti generované podobě tyto změny zachovat, navrhnout jejich začlenění do CASE nástroje, příp. upozornit, že nejsou konzistentní.

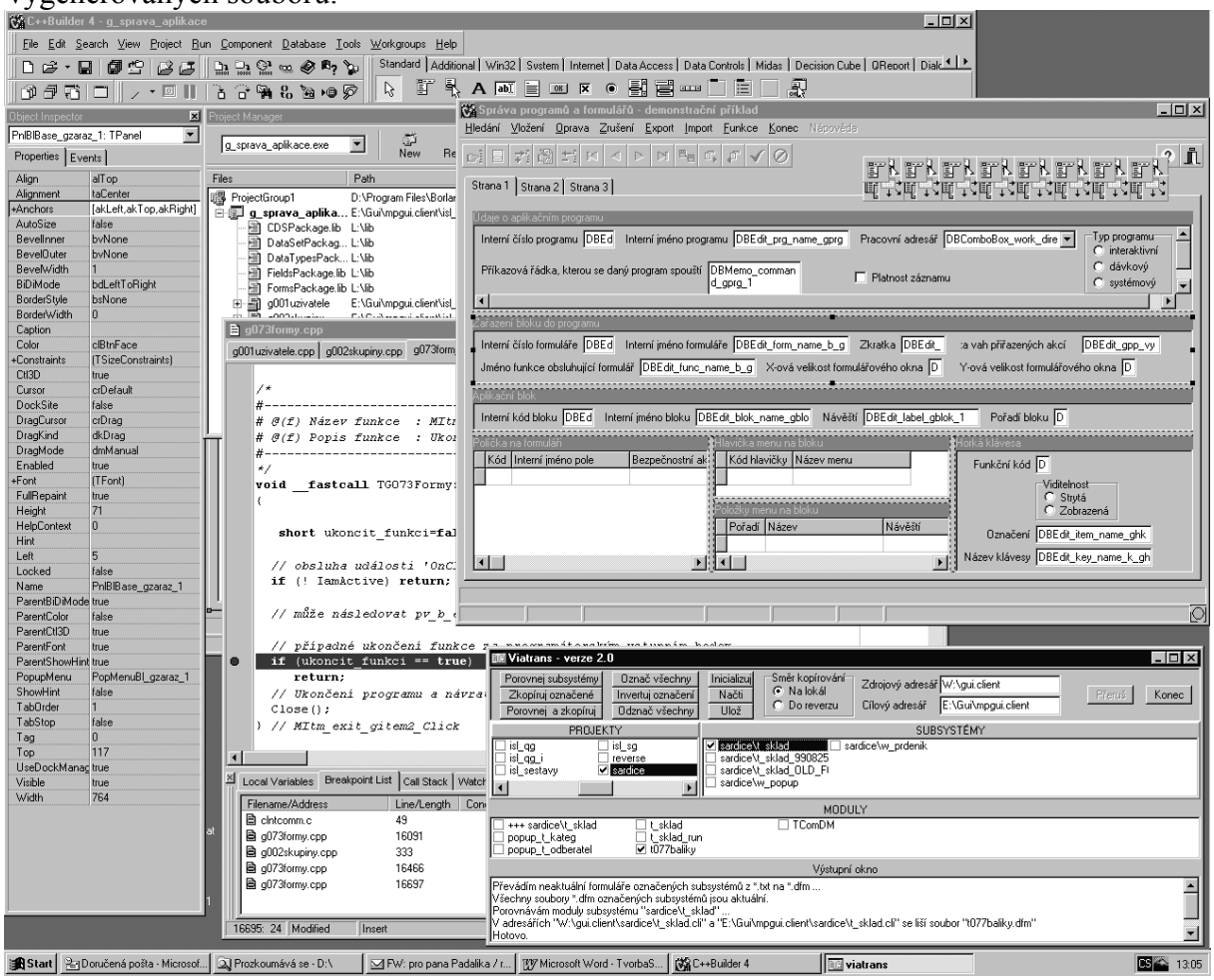
## 5. Vývojové prostředí a schéma tvorby aplikací

Pro lepší představu o použité technologii jsou na následujících obrázcích zachycena vývojová prostředí, ve kterých probíhá vývoj aplikací.



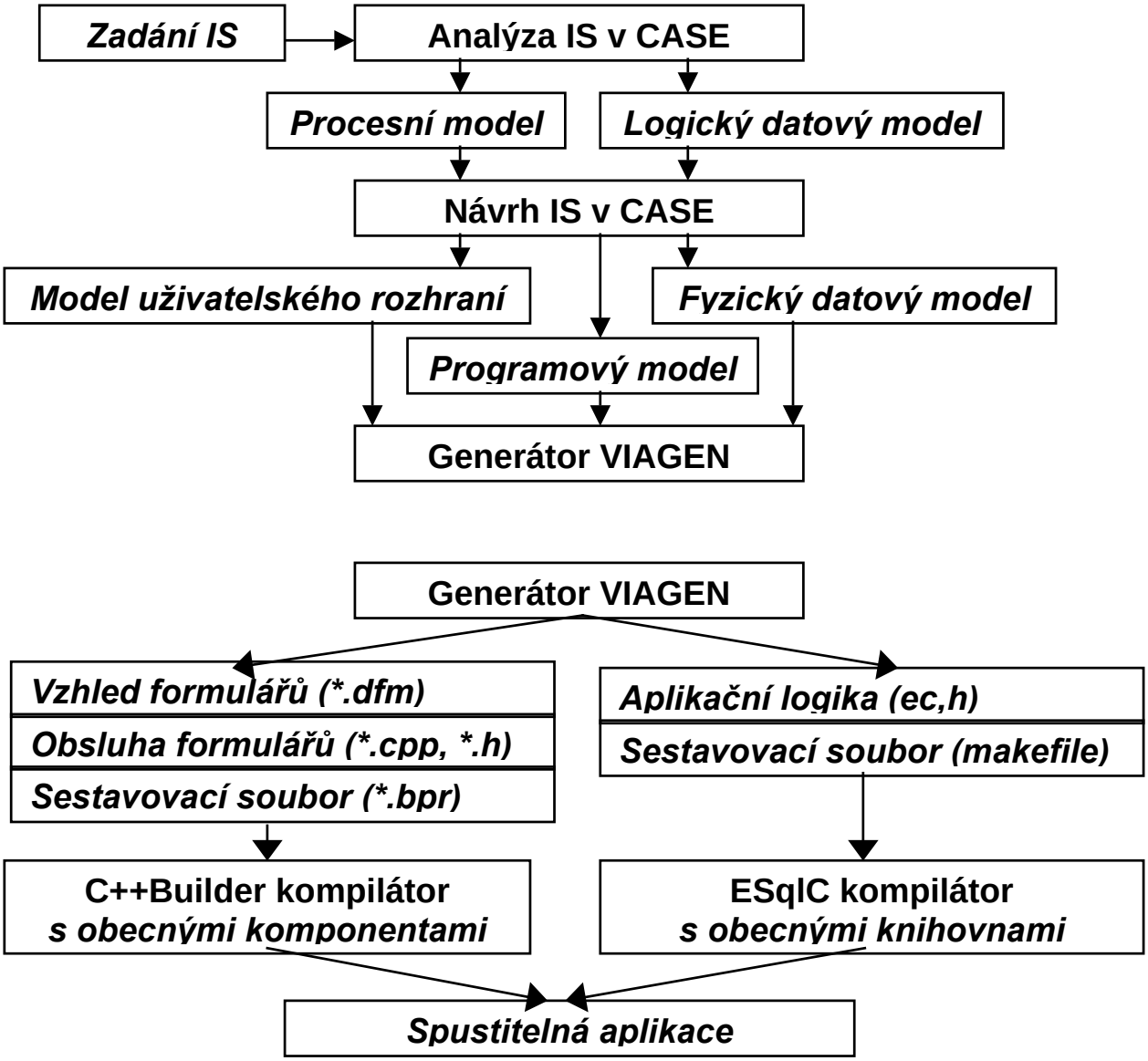

Obr. 4: Ukázka vývojového prostředí VantageTeam v X-Window

Na obrázku 4 je zachyceno vývojové prostředí CASE VantageTeam v X-Window, ve kterém probíhá analýza a návrh aplikací a finální tvorba aplikační vrstvy. Jsou zde zobrazena čtyři okna VantageTeamu, přičemž vlevo nahoře je okno obsahující seznam vyvíjených subsystémů (logických celků aplikace), vlevo dole je otevřený diagram s modelem obsahu vyvíjeného formuláře, vpravo nahoře je okno se seznamem vygenerovaných souborů pro aplikační i prezentační vrstvu a vlevo dole je okno zobrazující obsah jednoho z vygenerovaných souborů.




Obr. 5: Ukázka vývojového prostředí C++Builderu v MS Windows

Na obrázku 5 je zachyceno objektově orientované vývojové prostředí C++Builderu v MS Windows, ve kterém probíhá vývoj komponent a provádí se kompilace a finální úpravy klientské části aplikace. V levé části obrázku je zobrazeno okno s vlastnostmi a metodami objektů, uprostřed je okno se seznamem komponent a vygenerovaných souborů a pod ním je okno, ve kterém probíhá ladění konkrétní metody. Vpravo nahoře je formulář s vizuálními objekty pro finální úpravu uživatelského rozhraní a vpravo dole je okno programu Viatrans, který slouží pro automatickou transformaci souborů a objektů mezi jednotlivými vývojovými prostředími.




Uvedené schéma demonstruje způsob tvorby aplikací od úvodního zadání informačního systému po vytvoření spustitelné provozuschopné aplikace. Základním písmem jsou zobrazeny fáze projektu a použité nástroje, kurzívou jsou zobrazeny vstupy a výstupy, které během jednotlivých fází vznikají. Schéma je značně zjednodušené a nemůže zdaleka pokrýt celý vývojový cyklus tvorby IS, pro základní představu však postačuje.

**6. Závěr**

Dlouhou dobu sloužily CASE nástroje pouze na analýzu aplikace bez možnosti jejich využití pro automatizovanou tvorbu programů. Poslední doba si vynucuje stále větší automatizaci jednotlivých fází vývoje informačních systémů. Prezentovaná technologie je důkazem toho, že i fázi programování a návrhu je možné zautomatizovat a zužitkovat tak výsledek práce analytiků v CASE nástroji. Není přitom nutné zabývat se rutinními činnostmi ve fázi návrhu a programování a je možné maximálně zaměřit pozornost na požadavky zákazníka a na důkladnou analýzu vyvíjeného systému.

Nasazení technologie popisované v příspěvku do praxe prokázalo, že s jejím využitím lze rychle a efektivně vyvíjet i značně rozsáhlé informační systémy. Použití technologie vedlo ke zrychlení a zlevnění vývoje, zkvalitnění výsledných aplikací, snížení jejich chybovosti a zlepšení jejich udržovatelnosti.

## Literatura


1. Firemní materiály firmy AURA, s.r.o., Brno, 2000
2. Horstmann, C.S.: Vyšší škola objektového návrhu v C++, Vydavatelství SCIENCE, Veletiny, 1997.
3. Straka, M.: Vývoj databázových aplikací. Edice Nestůjte za dveřmi. Vydavatelství Grada, Praha, 1992.
4. Buřival, Z.: Řízení velkých softwarových projektů, Systémová integrace '98 – Sborník, Praha, 1998
5. Cayenne Software, Inc., VantageTeam User Interface Guide, Burlington, MA USA, 1997

Virius, M.: C++Builder 4.0, Grada Publishing, 1999