ŘÍZENÍ OBJEKTOVĚ ORIENTOVANÉ ANALÝZY A DESIGNU

Vladimír Zyka

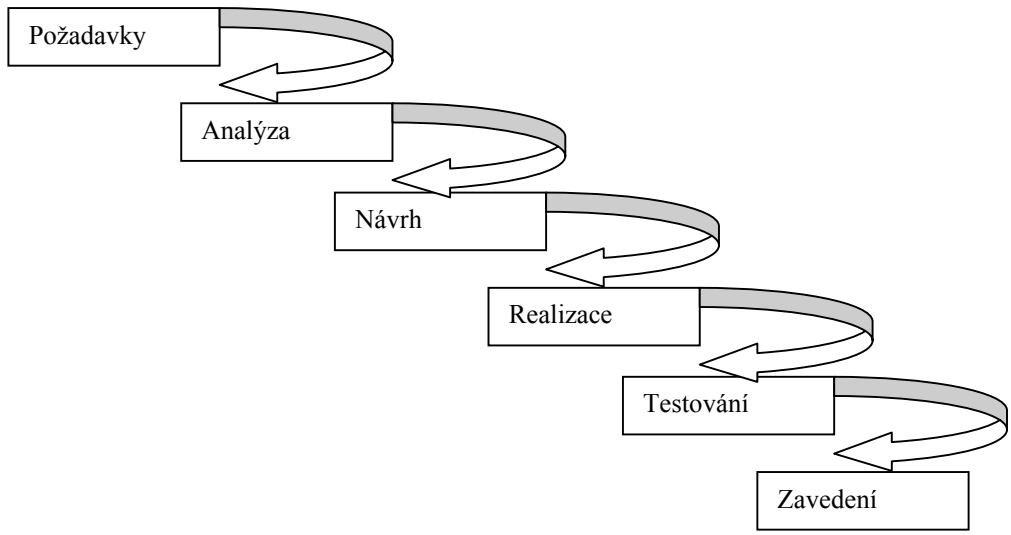
ITS a.s., Vinohradská 184, 130 52 Praha 3, ČR

1. Něco o historii v řízení projektování informačních systémů

Během mé 15-ti leté praxe v oblasti návrhů a realizací informačních systémů jsem prošel třemi vývojovými etapami řízení projekčních a programátorských činností. Každá z těchto etap byla determinována vždy úrovní technického a programového vybavení, jenž se v té které etapě mohla k těmto činnostem použít. Prudký proces zdokonalování technického a programového vybavení bezpochyby zásadně ovlivňuje činnosti spojené s řízením návrhu a vývoje informačních systémů. I přes tento dominantní faktor ovlivňující úroveň řízení projektů nelze přehlédnout další faktory, které způsobily zcela zásadní změny v pohledu na činnosti vykonávané v rámci projektování informačních systémů.

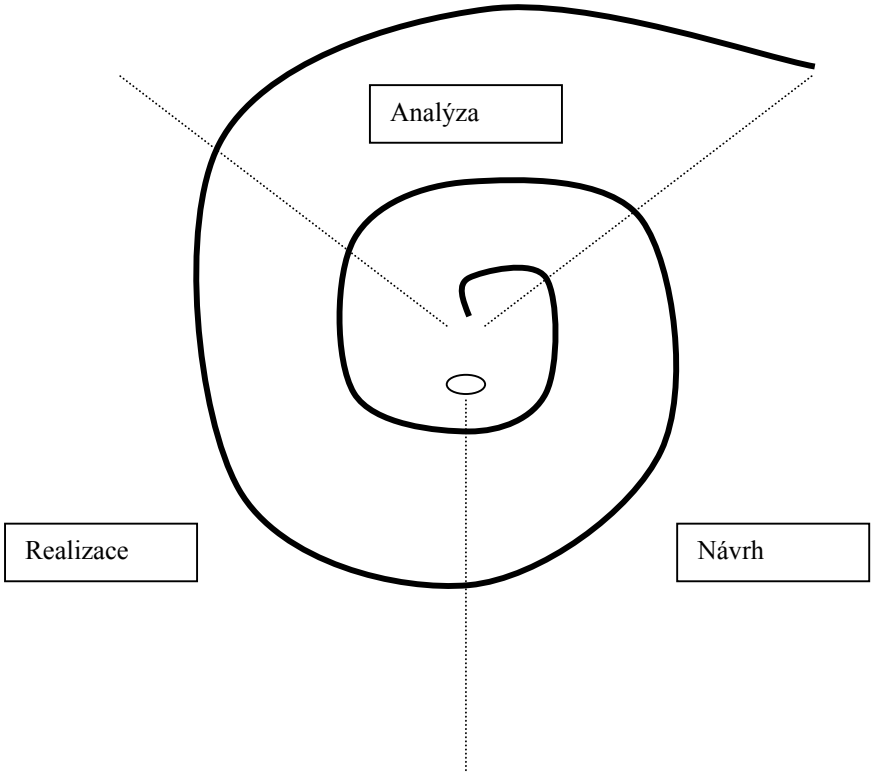
Díky nástupu objektových technologií a metodologií došlo k zcela odlišnému chápání životního cyklu projektu. Namísto tzv. vodopádového životního cyklu projektu, jenž byl představován jednotlivými po sobě následujícími fázemi (Požadavky-Analýza-Návrh-Realizace-Testování-Zavedení) došlo k prosazení tzv. iterativního životního cyklu viz [2]. Iterativní životní cyklus mnohem lépe odráží činnosti skutečně prováděné v průběhu životního cyklu projektu. Zásadní nevýhodou vodopádového životního cyklu byla skutečnost, že se projekt nemohl efektivně přizpůsobovat novým technologiím a potřebám uživatelů. A tak jednou provedená analýza a design určitého řešení nešla jednoduchým způsobem rozvíjet a zdokonalovat.

Obr.1: vodopádový životní cyklus projektu



Iterativní cyklus projektu, předpokládá opakovaný proces „Analýza-Návrh-Realizace-Testování-Zavedení“ s ohledem na maximální využití všech dosud provedených činností v předchozí iteraci. Znamená to tedy, že takto chápaný životní cyklus projektu umožňuje jeho stálý evoluční rozvoj. Výhody tohoto faktoru jsou nevyčíslitelné a snižují zcela zásadně náklady softwarových společností spojené s dalším vývojem informačních systémů.

Obr.2: iterativní (spirálový) životní cyklus projektu



Samotné přihlášení se k iterativnímu cyklu řízení projektu a využívání moderních objektově-orientovaných technologií však nestačí. Je naprosto nezbytné stanovit si a uvést v život detailně propracovanou metodologii podporující tento model životního cyklu projektu. Stanovení metodologie podporující iterativní životní cyklus projektu představuje:

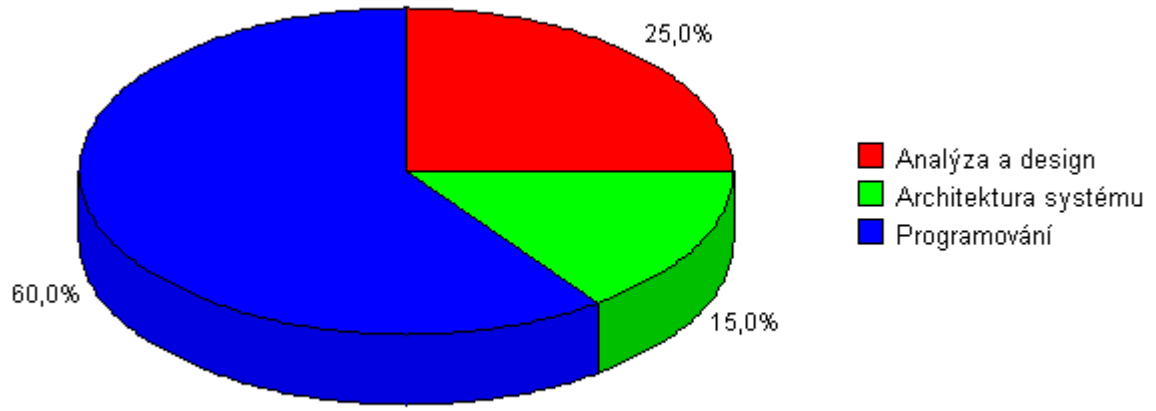
- ✓ přesně definovaný souhrn činností a rolí členů v projekčního týmu
- ✓ souhrn a návaznosti jednotlivých technologických nástrojů podporující jednotlivé činnosti
- ✓ přísné programové konvence
- ✓ definici vnitřní architektury a společné infrastruktury projektu

Takto propracovaná metodologie je alternativně nazývaná jako „Application Framework“ [4]. Podle toho, jak kvalitně je navržen „Application Framework“, lze očekávat v budoucnu výhody vyplývající z iterativního modelu životního cyklu projektu. Je třeba konstatovat, že tvorba takovéhoto aplikačního frameworku je nesmírně obtížná a důležitá činnost předcházející prvnímu reálnému projektu. Bohužel dnes existuje celá řada aplikačních frameworků a každý má své výhody ale i nevýhody. Neexistuje tedy universální lék na trvalé a optimální řešení v řízení projekčních činností. Naopak existuje mnoho cest, jak řídit projekt s nejmodernějšími objektově-orientovanými nástroji, ale jen některé vedou k vysněnému cíli.

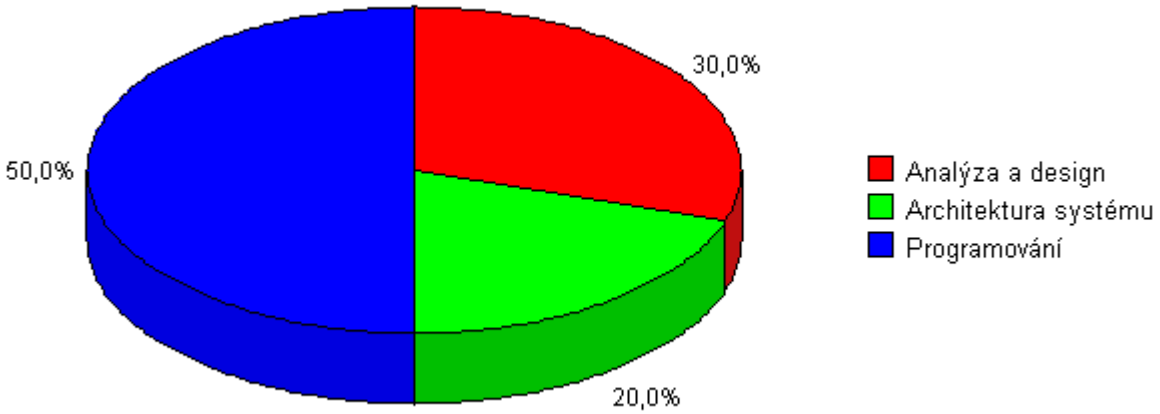
Obr.3: činnosti v historických etapách řízení projektů

Činnost	3GL	4GL a RAD	OOAD
Iterativní (evoluční) vývoj projektu	*	**	*****
Specializace činností a personální zaměnitelnost členů týmu	*	**	****
Autodokumentace projektu	**1/2***	**	****
Požadovaná kvalifikace architekta projektu	***	***	*****
Požadovaná kvalifikace analytika	***	***	*****
Požadovaná kvalifikace programátora	***	**	*

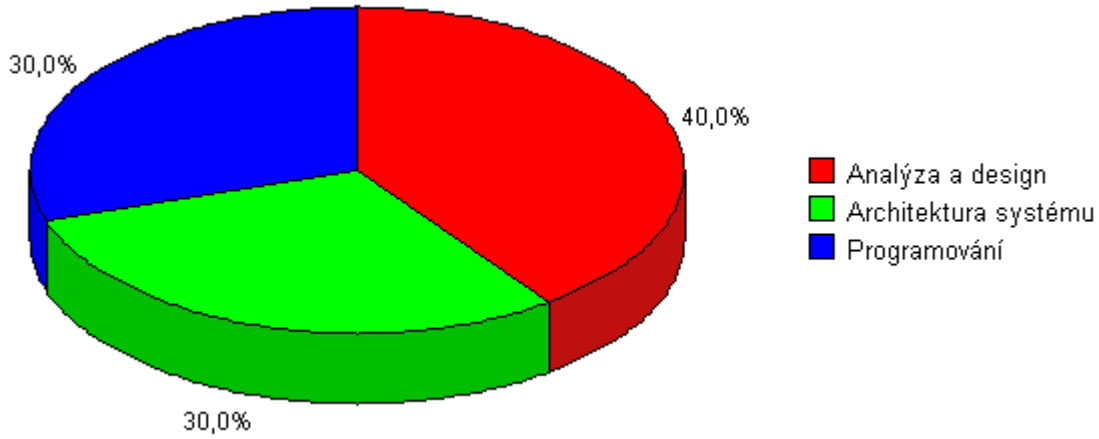
Obr. 4: podíl činností v období 3GL



Obr. 5: podíl činností v období 4GL a RAD



Obr.6: podíl činností v současnosti



2. Rational Rose for Java – efektivní a přehledný nástroj pro návrh objektových modelů

Nástroj Rational Rose for Java je CASE nástroj pro objektové modelování [1]. Prostřednictvím tohoto nástroje lze vytvářet objektově-orientované modely. Tyto modely mohou zachycovat odraz reálného světa, ale i sofistikovaně popisovat softwareový produkt. Lze ho tedy efektivně a smysluplně využít jak v oblasti analýzy reálných procesů tak i k jejich detailnímu počítačovému návrhu. Pro jednotlivé činnosti počínající analýzou problému a končící jejich detailní programovou specifikací existují v Rational Rose různé typy diagramů

a technik, jež proces „analýza-návrh“ podporují. Jednotlivé modely vytvářené v Rational Rose odpovídají notaci UML (Unified Modeling Language).

Zásadní výhody ve využívání Rational Rose v našem aplikačním frameworku jSPEC spatřuji v následujícím:

- ✓ podpora a dokumentace iterativního procesu „analýza-návrh-programování“
- ✓ vizualizace podstatných charakteristik problémových oblastí
- ✓ přesná programová specifikace (jednoznačný interface mezi analytikem a programátorem)
- ✓ dobré podmínky pro udržení jednotné architektury systému
- ✓ podpora metodologie Roundtrip Reengeering

Díky využití Rational Rose se v našem projekčním oddělení otevřela cesta ke spolupráci s externími programátory. Prostřednictvím Rational Rose lze velice exaktně definovat jednotlivé třídy, jejich atributy a metody a asociace mezi jednotlivými třídami. Takovéto zadání již nevyžaduje žádnou další specifikaci a výstupem z Rational Rose je skelet navržené třídy ve formě zdrojového kódu v jazyce Java. Zdrojové kódy obsahují programovou dokumentaci vytvořenou analytikem ve formě dokumentačních řádků JAVADOC. Jediným úkolem programátora se tak stává implementace kódu realizující jednotlivé metody. Těla jednotlivých metod obsahují kód, jenž by měl splňovat programové konvence. Díky sofistikované specifikaci jednotlivých tříd a jejich asociací, popř. generalizací, lze velice jednoduše udržovat „čistotu architektury“ daného softwareového produktu.

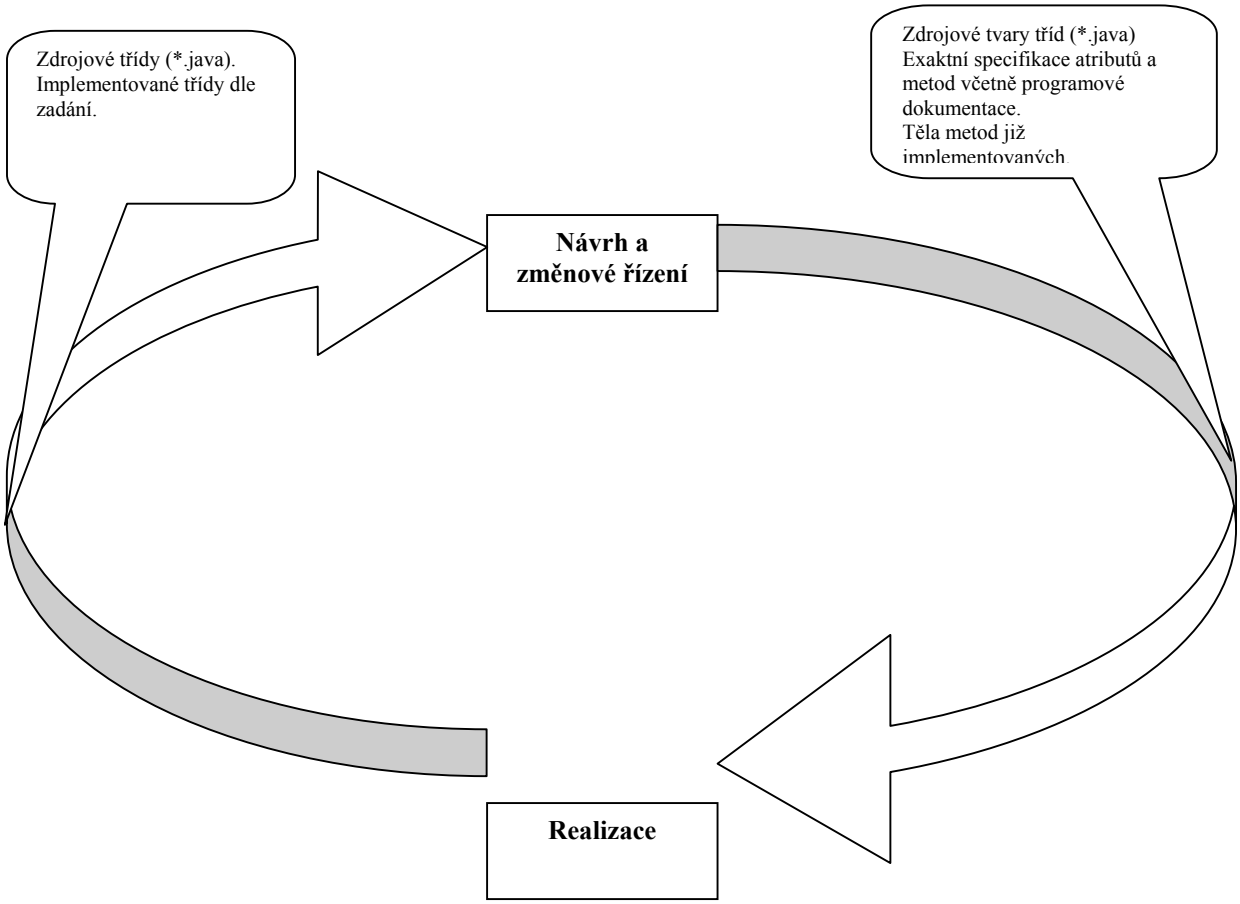
Rational Rose však nedává žádný návod, jak stanovit architekturu projektu. Stanovení architektury projektu představuje složitý proces vytvoření technologických nástrojů, abstraktních tříd, rozhraní a pomocných tříd, zajišťující efektivní návrh a realizaci vlastního projektu. V etapě využívání vizuálních generátorů aplikací, 4GL a RAD nástrojů byla architektura projektu determinována možnostmi daného prostředí a vývojové prostředí mnohdy samo vytvářelo architekturu projektu.

V případě použití nástrojů Rational Rose a Visual Age for Java je možno se vydat nejrůznějšími cestami a vyrobit si různé vlastní architektury projektu nebo použít již vytvořené architektury třetích subjektů. Společnost ITS a.s. vyvinula aplikační framework jSPEC, jehož součástí je i souhrn tříd, komponent a nástrojů zabezpečující tvorbu moderní vícevrstvé architektury projektu. Všechny součásti architektury jSPEC jsou navrženy a dokumentovány právě prostřednictvím Rational Rose a Visual Age for Java.

Analýza a návrh jednotlivých tříd a jejich programová realizace je prováděna specializovanými členy projekčního týmu. Exaktní rozhraní mezi těmito činnostmi umožňuje oddělit od sebe analytické a návrhářské činnosti od vlastního programování a to i personálně. Ve struktuře řešitelského týmu dochází k posilování významu a nároků na analytiky a návrháře systému a do pozadí se dostávají vlastní programátoři (např. externí spolupracovníci).

Rozhraní mezi nástroji Rational Rose a Visual Age for Java je realizováno přímo prostřednictvím zdrojových kódů jednotlivých tříd. Toto rozhraní je navrženo tak, aby byl podporován iterativní životní cyklus projektu. Spolupráce mezi nástroji Rational Rose a Visual Age for Java podporuje metodologii tzv. Roundtrip Reengeering, jenž umožňuje v jednotlivých iteracích provádět předávání zdrojových kódu z Rational Rose do Visual Age for Java a opačně zdrojový kód doplněný o implementaci jednotlivých metod programátory zpět do Rational Rose (po zpětném importu do Rational Rose, obsahuje objektový model i implementaci metod a při dalším exportu do Visual Age for Java tato implementace zůstává).

Obr. 7: Schéma procesu „Roundtrip reengeering“



3. Visual Age for Java – výkonný nástroj pro programovou realizaci , testování a řízení verzí objektových modelů

Visual Age for Java je jedním z rodiny vizuální vývojových nástrojů společnosti IBM [3]. Nástroj umožňuje programovou realizaci jednotlivých tříd v jazyce Java. Podporuje tvorbu tzv. Java Beans, javovských tříd (komponent) optimalizovaných pro vizuální návrh aplikačních programů. Součástí nástroje je souhrn hotových projektů obsahující řadu technologických tříd, Java Beans a nástrojů pro rychlý a efektivní návrh aplikací.

Visual Age for Java podporuje návrh i následnou implementaci EJB (enterprise java beans).

Součástí nástroje je velice efektivní debugger. Integrální součástí nástroje je objektový repozitář jednotlivých tříd, umožňující elegantní křížovou manipulaci s jednotlivými třídami.

Podstatnou součástí repozitáře je systém pro řízení verzí jednotlivých objektů. Jednotlivé křížové operace lze provádět i napříč jednotlivými verzemi!

Uživatelem tohoto nástroje jsou v našem týmu především programátoři. Nelze však říci, že by tento nástroj nepoužívali i analytici a návrháři. Jednou z nevýhod Rational Rose je skutečnost, že nelze provádět vizuální návrh uživatelských rozhraní. Tuto činnost v našem projekčním týmu řeší analytici a návrháři právě prostřednictvím Visual Age for Java.

Samozřejmým předpokladem každého analytika a návrháře našeho projekčního týmu, je znalost JDK a dalších standardních tříd vyplývajících z použité architektury. V předchozích

etapách řízení projektů byla využívána funkčnost tehdejších vývojových prostředí popř. knihovny vlastních podprogramů a modulů. Četnost těchto modulů a podprogramů byla řádově nižší než četnost balíků standardních tříd a komponent volně dostupných a rozšiřujících JDK a to i od třetích subjektů.

V oblasti jazyka Java jsou dnes k dispozici tisíce volně dostupných tříd řešících obecné problémy a každým dnem dochází k vzniku dalších a dalších tříd a balíků. Z tohoto důvodu jsme v našem řešitelském týmu ustanovili funkci vyhledávače takovýchto obecných tříd, jehož náplní je posuzovat možnou integraci takovýchto tříd do projektu.

Lze si představit značné nároky na faktografickou znalost existujících tříd a balíků, jež je předpokladem efektivního návrhu řešení projektu.

4. jSPEC – aplikační framework pro iterativní řízení objektově-orientované analýzy a designu

jSPEC je aplikační framework, vyvinutý společností ITS a.s. Je určen k iterativnímu procesu analýza-návrh-realizace aplikací podporujících automatizaci interních vnitropodnikových procesů (multi tiers architektura) nebo extranetových e-businessových aplikací typu B2B a B2C (server side Java).

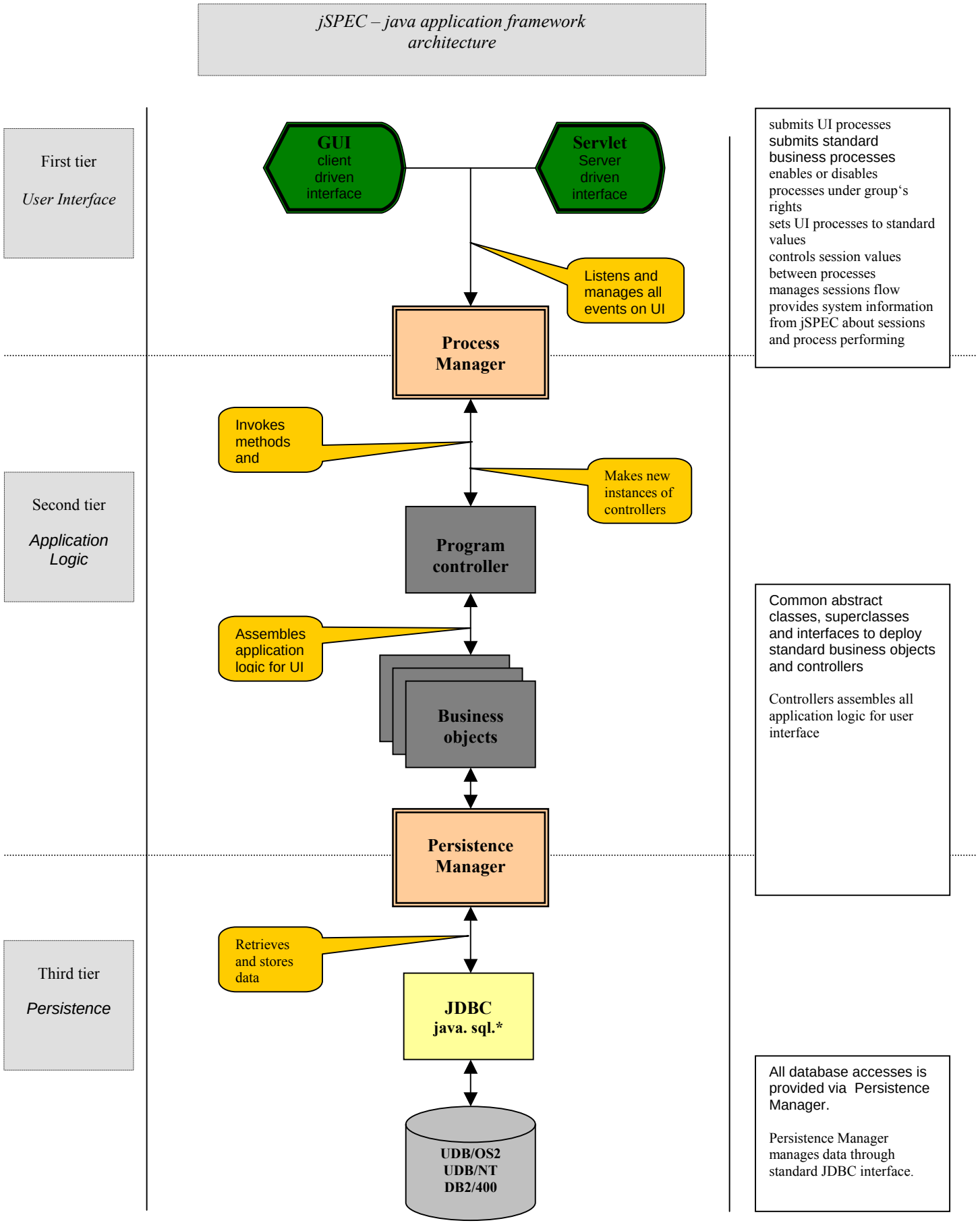
jSPEC představuje souhrn nástrojů, metodik, softwarové architektury projektu, struktury projekčního týmu a jednotlivých rolí v projekčním týmu a zásad a konvencí zabezpečujících spolupráci jednotlivých členů týmu.

Aplikace vytvořené v rámci aplikačního frameworku jSPEC je portabilní z pohledu serverů na nejrozšířenější serverové operační systémy (OS/400, WINDOWS/NT, OS/2, AIX).

Aplikace realizované vícevrstvou architekturou jsou plně ovládány prostřednictvím komponent grafického uživatelského rozhraní a mohou být tudíž provozovány na stanicích pracujících pod JVM (Java Virtual Machine).

E-businessové aplikace typu B2B a B2C jsou ovládány výlučně prostřednictvím html prohlížečů a komunikují s aplikačním serverem výhradně prostřednictvím protokolu http.

Obr.8: architektura aplikačního framworku jSPEC



Takovéto aplikace jsou provozovatelné na tzv. tenkém klientu včetně IBM Network Station (nevyžadují pracovní stanici Java Virtual Machine).

Vícevrstvá architektura tvorby projektů umožňuje koncentrovat aplikační logiku do tzv. business objektů a programových kontrolerů. Tyto třídy jsou navrženy jako implementačně nezávislé. Veškeré databázové zpracování (perzistence objektů) je realizováno prostřednictvím samostatné vrstvy. Analogicky je řešeno rozhraní mezi uživatelským rozhraním a těmito objekty. Vazbu mezi UI (user interface) a aplikační logikou realizuje Process Manager, jenž obsahuje funkce transakčního monitoru.

Process Manager zpřístupňuje a naslouchá všem událostem uživatelského rozhraní (GUI i html) a na základě těchto událostí přiděluje jednotlivým relacím instance příslušných programových kontrolerů a spouští příslušné procesy (metody programových kontrolerů).

Každý programový kontroler extenduje obecnou funkcionalitu objektu typu kontroler (obsahuje obecně platné metody) a implementuje interface. Implementace společného interface do jednotlivých programových kontrolerů unifikuje a standardizuje jednotlivé programové kontrolery. Takovýto proces standardizace jednotlivých typů programových kontrolerů vnáší značnou přehlednost a umožňuje nezávislost programového kontroleru na jeho autorovi.

Veškeré výkonné operace s třídami typu entity (získávání a ukládání dat z datových skladů a manipulace s takovýmito daty ve vnitřní paměti počítače) realizují třídy typu business object. Programové kontrolery se výhradně odkazují na metody tříd typu business object. Tato praktika umožňuje vysoký stupeň nezávislosti programových kontrolerů na fyzické struktuře datových skladů (implementačním datovém modelu). Každá třída typu business object má svoji genezi a standardně implementuje společný interface.

Vrstva perzistence objektů realizuje metody získávání a ukládání dat z datových skladů. Všechny business objekty využívají k I-O operacím s daty výlučně vrstvu perzistence. Persistence využívá vlastního popisu datových elementů. Tato skutečnost umožňuje nezávisle na vrstvě business objektů provádět změny v implementaci datového modelu (např. záměny dataelementů mezi jednotlivými databázovými tabulkami).

5. Závěr

Neustále se zrychlující vývoj informačních technologií nutí softwareové společnosti k hledání základních stavebních kamenů, které přežijí a budou využitelné i za několik málo let v době, kterou dnes nedokážeme přesně odhadnout.

Na obzoru se smráká i klasickým „komplexním“ sotwareovým produktům řešícím vnitropodnikový informační systém, neboť řada procesů, jenž jsou těmito balíky realizovány, ztratí svůj význam. Informační systémy jednotlivých společností budou čím dál tím více orientovány na kolaborující subjekty s cílem poskytnout těmto subjektům potřebné automatizované transakce pro získávání a předávání informací mezi těmito subjekty.

Standardem elektronické výměny informací se v krátké době stane formát XML se standardizovaným slovníkem dataelementů.

Pohledy do vnitřku informačního systému daného subjektu nebudou zprostředkovávány zaměstnanci dané organizace (vybavenými řadou vnitropodnikových automatizovaných transakcí), nýbrž budou realizovány extranetovými transakcemi typu B2C a B2B.

Zcela jistě však budou vznikat nové transakce určené pro zaměstnance dané organizace, jenž budou sloužit ke kontrole a modifikaci průběhu zpracování. Lze předpokládat, že zcela zaniknou nebo budou věcně transformovány celé subsystémy, které dnes hrají v informačních

systémech organizací klíčové role (řízení odbytu, fakturace, evidence došlých faktur, zpracování bankovních transakcí v dané organizaci atd.).

Tyto skutečnosti budou mít za následek značnou poptávku po inovaci stávajících informačních systémů a budou tak generovat mnoho příležitostí pro softwareové společnosti.

V souvislosti s očekávanou mohutnou inovací informačních systémů je nezbytné vytvořit si uvnitř vývojového oddělení určitý aplikační framework podporující iterativní životní cyklus projektu s cílem „odolávat“ bouřlivému vývoji informačních technologií a schopnosti absorbovat do svého řešení nové trendy a technologie s minimálními náklady.

Literatura

1. Rational Rose – Using Rational Rose 98. Rational Software – Rose Documentation
2. SVAČINA, J. Školení Rational Rose 98. UNICORN Education s. r. o. 1998
3. IBM Visual Age for Java – User Guide. IBM 1999.
4. DOUGLAS, L.,LITLLE, M., ZULIANI, F. San Francisco Evaluation Kit (V1R2) – Getting Started. IBM 1999.
5. San Francisco Concepts & Facilities. IBM 1998.