Architektura programových systémů

Jan M Honzík

Ústav informatiky a výpočetní techniky, Fakulta elektrotechniky a informatiky VUT v Brně, Božetěchova 2, 612 66 Brno 12

Abstrakt

Tvůrci softwaru používají programových vzorů smysluplně ale často podvědomě. Neformální způsob popisu systému vede k programátorské tvořivosti bez výrazné přesnosti, koncepčnosti a konzistence. Rozsah a složitost programových systémů se stále zvětšuje a proto návrh a specifikace (popis) celého systému začíná být významnější, než volba algoritmů a datových struktur. To vytváří podmínky pro nově se formující disciplínu zvanou Architektura programových systémů (Software Architecture).

1. Architektura programových systémů (Software Architecture)

Architektura programových systémů (APS) se zabývá popisem elementů pro výstavbu systémů, interakcí mezi těmito elementy, vzory a jejich skladbou a omezeními těchto vzorů. Každý systém může vystupovat jako element rozsáhlejšího systému. Současný stav neformálnosti návrhu abstrakcí na nejvyšší úrovni zdánlivě naznačuje, že systematičnost návrhu APS nemá pro softwarové inženýrství velký smysl. Ze dvou důvodů tomu tak není:

- Tvůrci vyvinuli a používají slovník pojmů a stylů, který je dostatečně bohatý a společný velké komunitě návrhářů.
- Přesto, že architektonické struktury jsou abstraktní ve vztahu k výpočetním detailům, vytvářejí přirozený rámec pro porozumění širším systémovým vztahům, jako jsou globální toky dat, vzory komunikace, provádění řídicích struktur, měření a dimenzování. APS se snaží definovat systém v pojmech výpočetních komponent a jejich vztahů. Komponentami jsou celky na úrovni klientů a serverů, databází, filtrů a vrstev hierarchického systému. Interakci na této úrovni reprezentuje volání procedury nebo přístup ke společné proměnné, ale také složitější vazby jako protokol klient-server, protokol přístupu do databáze, asynchronní zpracování více událostí a nebo datové proudy (streams) zpracovávaného souboru.

Návrh rozsáhlého systému můžeme zhruba rozdělit do tří až čtyř úrovní:

- Na nejvyšší úrovni je to architektura systému, kde komponenty jsou moduly, propojení modulů nebo subsystémů a existují operátory pro propojení subsystémů
- Program, kde komponenty jsou prvky programovacího jazyka jako příkazy, data a

kompoziční principy jako záznam, pole nebo procedura
- Úroveň stroje, kde komponenty jsou paměťová místa, volání zásobníku, registry a kompoziční principy jsou popsány prostředky strojového kódu.
- Nejnižší úroveň je popsána ryze hardwarovými pojmy na úrovni zesilovačů klopných obvodů, hradel, a jiných aktivních i pasivních elektronických komponent.

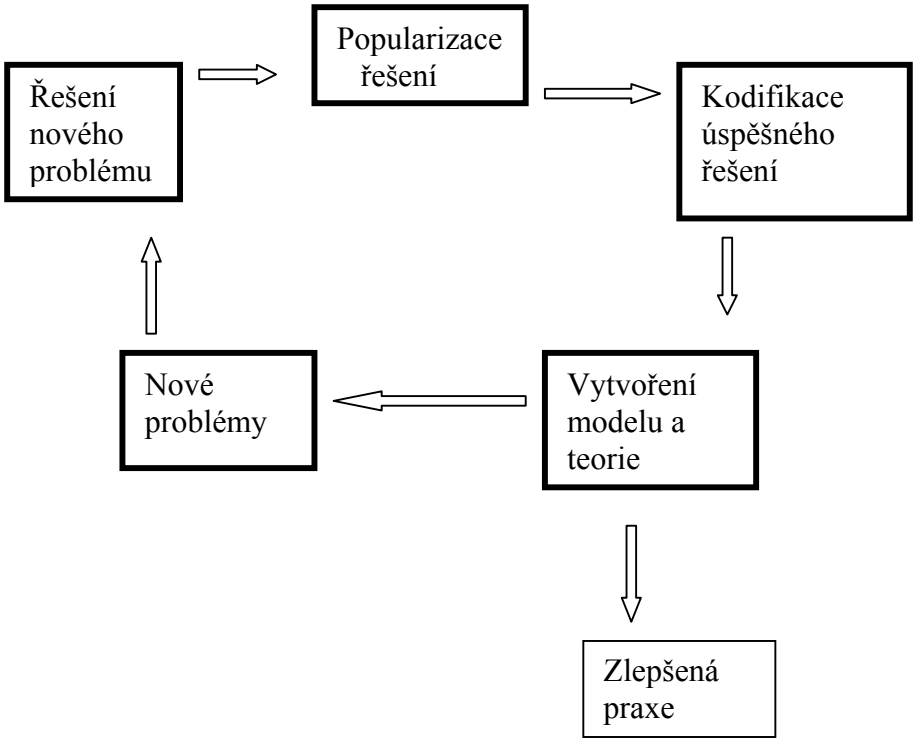
1.1 Inženýrská disciplina pro programování

"Softwarové inženýrství" je pojem, který vznikl v r.1968 na semináři NATO o perspektivách tvorby programů. Co je to "inženýrství"? Je to proces vytváření ekonomicky účinných řešení praktických problémů při využití současných vědeckých poznatků, pro produkci věcí, které slouží společnosti v tržně zbožním vztahu.

Inženýrské návrhy lze nejsnadněji rozdělit podle míry originality a novosti návrhu na rutinní a inovativní. Rutinní návrhy využívají zkušenosti i prvky známých, úspěšně řešených problémů. Podporují je nejrůznější příručky, návody a popisy technologických postupů. Inovativní návrhy řeší novým způsobem nejčastěji málo známé nebo zcela nové problémy. Inženýrská praxe vyžaduje mnohem více rutinních řešení než inovativních. V oblasti software je výskyt inovativních návrhů častější než v jiných technických disciplínách. Je to dáno charakterem práce i charakterem programátorů a také historií vývoje disciplíny, v níž se akcentoval tvůrčí přístup před rutinou. Cesta k vyšší produktivitě však vede ke klasifikaci problémů a k tvorbě standardních řešení a nástrojů pro jejich podporu.

1.2 Východiska softwarové architektury

Výzkum v oblasti architektury programových systémů se začal formovat v prostředí softwarového inženýrství v 90. letech. Zatím jsou známa a pojmenována mnohá architektonická paradigmata (jako roury, vrstvy,klient-server), která jsou ale vnímána více-méně zhruba principiálně a většinou se implementují ad hoc stylem. V poslední době byly vymezeny hlavní oblasti a směry dalšího vývoje APS. Jsou to:
- Jazyky pro popis architektur. Jde o vývoj jazyka usnadňujícího komunikaci mezi tvůrci programových systémů
- Kodifikace architektonických zkušeností. Jde o nacházení taxonomie a vytváření katalogizace nejrůznějších architektonických principů a vzorů vytvořených při vývoji programových systémů.
- Rámce specifických domén. Výsledkem výzkumu v této oblasti je hledání a vymezení architektonických rámců pro specifické třídy programových systémů jako jsou např. letecké (avionické) řídicí systémy, robotika, rozhraní člověk-počítač apod. Vymezení může vést k tvorbě specializovaných podpůrných programovacích nástrojů, zvyšujících produktivitu tvorby programových systémů.



Obr. 1.1 Cyklus vývoje technologie a teorie

2. Architektonické styly

Dříve, než se začalo hovořit o disciplinární oblasti nazývané APS, se již běžně používaly pojmy jako "systém klient-server", návrh typu "roura-filtr" nebo "vrstvová struktura". Jako další přívlastky organizace systému jsou známy např. "objektově orientovaná organizace", nebo "řízení tokem dat". Ačkoliv taxonomie v této oblasti zdaleka ještě není uzavřena, některé známé kategorie lze shrnout do těchto skupin:

- Systémy toku dat
  - Sekvenční dávkové systémy
  - Systémy s rourami a filtry
- Systémy volání a návratu
  - Hlavní program a podprogramy
  - 00 systémy
  - Hierarchické vrstvy
- Nezávislé komponenty
  - Komunikující systémy
  - Systémy s událostmi

- Virtuální stroje
  - Interprety
  - Systémy založené na pravidlech
- Datové systémy
  - Databáze
  - Hypertextové systémy
  - Tabule

Na nižší úrovni lze architekturu specifikovat jako soubor komponent propojený popsaným způsobem propojovacími prvky - *konektory*. Příkladem komponenty je např. filtr, vrstva apod. Příkladem konektoru je např. volání procedury, šíření zprávy apod.
Ke hledání a vymezení architektonického stylu pomohou odpovědi na takto položené otázky:

- Jaký je slovník návrhu, jaké jsou komponenty a konektory?
- Jaké jsou možné strukturální vzory?
- Jaký výpočetní model je základem návrhu ?
- Jaké jsou hlavní invarianty stylu ?
- Jaké jsou obecné příklady použití stylu ?
- Jaké jsou hlavní výhody a nevýhody použití stylu ?
- Jaké jsou typické specializace stylu ?

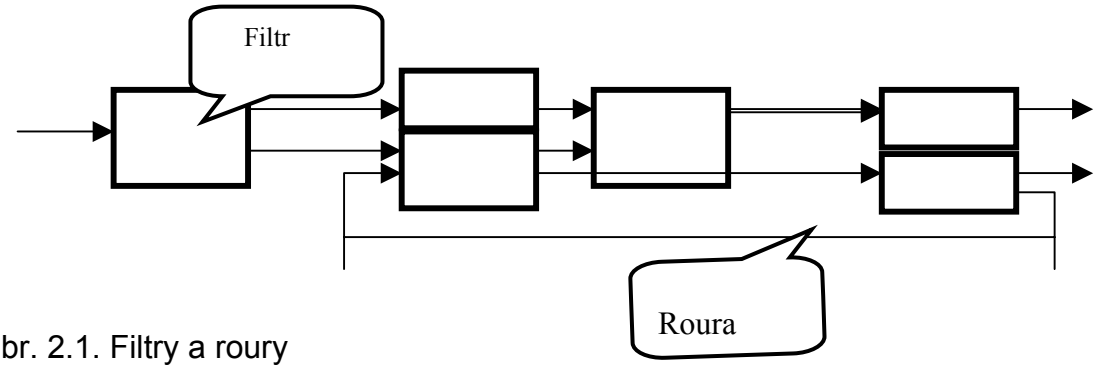
Mezi nejčastěji používanými styly jsou:

- roury a filtry
- objekty
- implicitní volání
- sklady (angl. repository -repozitáře)
- vrstvy
- interprety
- procesní řízení

2.1 Roury a filtry

Každá komponenta tohoto stylu má sadu vstupů a sadu výstupů. Komponenta čte proud vstupních dat na vstupech a produkuje proud výstupních dat na výstupech. Většinou jde o inkrementální transformaci, takže výstupní proud začne být vytvářen dříve, než skončí zpracování vstupního proudu. Proto se komponentě říká filtr. Konektory slouží jako spojovací zařízení, přenášející proud dat z výstupu jednoho filtru na vstup jiného filtru. Proto se zde konektorům říká roury (pipe, pipe-line). Mezi důležité invarianty stylu patří podmínka, že jednotlivé filtry jsou na sobě nezávislé entity, které nemají informaci o identitě předcházejících a následujících filtrů (neví od koho přijímají data a komu slouží jimi produkovaná data). Korektnost výstupu systému nesmí záviset na pořadí, v němž jednotlivé filtry provádí svou inkrementální transformaci. Obecnou specializací stylu mohou být: omezení topologie na lineárnístrukturu, omezení datové kapacity roury nebo typové omezení roury vyžadující pro přenos dobře definovaná a typově kontrolovaná data. Degenerovaný případ stylu zpracovává proud dat jako celek a tím se stává systémem s dávkovým zpracováním (batch processing). Nejznámějším případem tohoto stylu jsou programy jádra Unixu. Procesy Unixu (filtry) jsou propojeny run-time mechanizmem. Jiným příkladem je většina překladačů, z nichž řada pracuje inkrementálně. Systém filtrů a rour má řadu dobrých vlastností. Díky propojení filtrů s dobře definovaným chováním jsou dobře srozumitelné. Umožňují využít znovupoužití, protože kterékoli dva filtry jsou navzájem propojitelné. Usnadňují také vývoj - filtr může být nahrazen zlepšenou verzí se stejným chováním. Nevýhodou stylu je, že většinou míří k dávkovému stylu zpracování, a proto mohou být nepřijatelné pro interaktivní aplikace. Systém může přinášet potíže při zpracování vzájemných vazeb dvou oddělených, ale navzájem

závislých proudů dat.



Obr. 2.1. Filtry a roury

2.2 Objektově - orientovaný styl

Pro tento styl je charakteristické zapouzdření (enkapsulace) datových struktur a operací nad těmito daty do abstraktních datových typů nebo do objektů. Komponentami jsou objekty jim odpovídajících tříd. Objekty jsou typem komponent, kterým se říká "manažer", protože jsou zodpovědné za zachování integrity datového zdroje. Objekty vzájemně komunikují voláním metod (procedur a funkcí). Mezi invarianty patří zodpovědnost třídy za integritu datové reprezentace svých objektů a ukrytí reprezentace (neprůhlednost) před ostatními objekty. Existuje řada variací na OO styl, např. možnost, kdy objekty reprezentují paralelní úkoly, nebo mohou mít vícenásobná rozhraní.

Mezi výhody OO stylu patří neprůhlednost zapouzdření, která dovoluje změnit implementaci bez vlivu na klienty. Svázání dat a jejich operací do objektu dovoluje vytvářet návrh s využitím dekompozice problému na soubor vzájemně spolupracujících objektů - agentů. Nevýhodou OO stylu je, že pro vzájemnou komunikaci objektů (prostřednictvím volání metody) musí volající objekt znát identitu objektu, jehož metodu volá. Tím se výrazně liší od stylu "filtrů a rour". V modulárně orientovaných jazycích to znamená změnu v seznamech importovaných objektů všude, kde se vyskytuje změněný objekt.

2.3 Implicitní volání založené na událostech

V systémech, v nichž rozhraní komponent poskytuje soubor procedur a funkcí, jako v OO organizaci, probíhá interakce explicitním voláním jejich podprogramů. V poslední době vzrostl zájem o interakční techniku nazývanou "implicitní výzva" nebo "implicitní volání", "reaktivní integrace" nebo"selektivní vysílání".

Princip implicitního volání spočívá v tom, že místo volání procedury komponenta oznámí (vyšle) zprávu o jedné nebo více událostech. Ostatní komponenty mohou registrovat tuto zprávu a projevit o ni zájem aktivací vlastních procedur. Např. v integrovaném vývojovém programovacím systému, používaném pro vývoj a ladění programů, může událost způsobená dosažením kontrolního bodu (break point), která

zastaví ladicí chod programu, vyslat zprávu, jejímž důsledkem je nastavení editoru zdrojového textu do místa zastavení a zobrazení hodnot monitorovaných proměnných. Z architektonického pohledu jsou komponenty tohoto stylu moduly, jejichž rozhraní poskytuje jak sadu procedur, tak sadu událostí. Procedury mohou být aktivovány normálním způsobem, ale komponenta může aktivaci procedury spojit s výskytem jisté události v systému. Hlavním invariantem stylu je, že vysílající komponenta neví, které komponenty budou vyvolanou událostí ovlivněny. Komponenty nemohou předvídat pořadí zpracování, ani které procesy budou vyslanou zprávou aktivovány. Jako příklady použití lze uvést programovací nástroje integrovaných prostředí v databázových systémech, zajišťujících požadavek konzistence, nebo uživatelská rozhraní oddělující prezentaci dat od jejich zpracování. Výhodou stylu je snadné znovupoužití. Nová komponenta může být do systému uvedena tím, že se jí umožní reagovat na podněty systému. To umožňuje také vysokou flexibilitu a vývoj systému. Hlavní nevýhodou implicitního volání je, že se komponenty vzdávají řízení chodu systému. I když vyšlou podnět, nemají informaci o tom, kdo na podnět reagoval, a i když ho mohou zjistit, nemohou znát pořadí, v němž jiné komponenty reagovaly.

Dalším problémem je přenos dat mezi komponentami. Někdy lze přenos dat spojit s událostí, jindy se využívá společné globální paměti, což přináší některé problémy. Další nevýhodou tohoto stylu je obtížnost uplatnit tradiční postupy dokazování správnosti programu na základě počáteční a koncové podmínky (precedence a konsekvence).

2.4 Vrstvový systém

Vrstvový systém je hierarchické uspořádání, v němž každá vrstva poskytuje služby především pro sousední nadřazenou hierarchickou vrstvu a jako klient využívá služeb hierarchicky nižší vrstvy. Komponenta systému - vrstva - představuje virtuální stroj na určité úrovni abstrakce. Pravidlo striktní komunikace mezi vrstvami může mít různou úroveň volnosti.

Ve stylu vrstev jsou navrženy komunikační protokoly, v nichž nižší vrstvy řeší úkoly bližší strojové úrovni a nejnižší vrstvy realizují fyzické spojení. Jinými příklady mohou být databázové systémy nebo operační systémy. Mezi výhody vrstvového systému patří podpora návrhu se zvyšující se úrovní abstrakce. To umožňuje rozdělení složitého systému na menší části - vrstvy. Podobně jako u filtrů a rour podporuje tento styl striktnější komunikace mezi vrstvami záměny různých implementací téže vrstvy. Tato vlastnost vedla k normalizaci a tvorbě standardů (OSI-ISO model, X-Windows apod.). Nevýhodou může být, že ne všechny problémy lze snadno strukturovat do vrstev a potřeba většího výkonu může vést k nutnosti vyšší spřaženosti ne-sousedních vrstev. Někdy mohou nastat problémy i v oblasti standardizovaných problémů v oblasti komunikace, jako při mapování existujících protokolů do ISO rámců, protože protokoly zahrnovaly i několik vrstev.

2.5 Sklady - repozitáře

Ve stylu repozitář jsou dva odlišné typy komponent:

- centrální struktura dat reprezentující stav systému

- soubor nezávislých komponent pracující s centrálními daty

Podle zvolené strategie řízení se tento styl rozpadá na dvě základní kategorie:
a) Jestliže o výběru procesu rozhodne typ transakcí vstupního proudu, bude se systém chovat jako tradiční databáze.
b) Jestliže naopak o výběru dalšího procesu rozhodne stav centrálního repozitáře, jde o systém zvaný "tabule" (blackboard).

Systém tabule sestává ze tří hlavních částí:
- Znalostní zdroje - oddělené, nezávislé balíky aplikací znalostně orientovaných
- Struktura tabule - data o stavu problému organizovaná do aplikačně závislé hierarchie. Znalostní zdroje mohou měnit tabuli tak, že to postupně vede k řešení daného problému.
- Řízení je ovládáno plně stavem tabule. Znalostní zdroje se uplatňují, když jim to stav tabule umožní.

2.6 Interprety

Výsledkem organizace typu interpret je obvykle virtuální stroj. Interpret sestává z pseudoprogramů, které mají být interpretovány a z interpretačního stroje samotného. Pseudoprogram sestává z vlastního programu a z aktivačního záznamu reprezentujícího stav provádění interpretu. Interpretační stroj sestává z definice interpretu samého a ze stavu svého provádění. Interpret tedy sestává ze čtyř komponent:
- Interpretační stroj realizující vlastní práci
- Paměť, která obsahuje intepretovaný program
- Reprezentace stavu aktuálního řízení interpretačního stavu
- Reprezentace aktuálního stavu zpracovávaného (simulovaného) programu Interprety se často používají ke konstrukci virtuálních strojů, čímž vyplňují prostor mezi čistým programovým strojem a jeho sémantikou a výpočetním strojem realizovaným hardwarem

2.7 Procesní řízení

Další styl je založen na řídicí smyčce procesu. Tento systém není mezi programátory příliš známý, ale patří mezi základní styly. Na rozdíl od objektově orientovaného nebo funkčního návrhu, které jsou charakteristické různými druhy vyskytujících se komponent, návrh ve stylu řídicí smyčky je typický jak použitými komponentami, tak vztahy, které spojují komponenty pohromadě.

2.7.1 Řídicí paradigma

Spojité procesy mění vstupní veličiny na výstupní veličiny prováděním operací se vstupními a meziproduktovými veličinami. Hodnoty měřitelných vlastností stavu systému se nazývají proměnnými procesu. Proměnné procesu, které měří výstupní hodnoty, se nazývají řídicími proměnnými procesu. Vlastnosti vstupních hodnot, mezivýsledky a operace jsou zachyceny v ostatních procesních proměnných. Manipulovatelné proměnné jsou spojeny s těmi projevy, které mohou být změněny řídicím systémem za účelem regulování procesu. V řídicím paradigmatu se používají

pojmy z teorie řízení, jako:

- *Procesní proměnné* - vlastnosti procesu, které lze měřit;
- *Řídicí proměnná* - proměnná procesu, jejíž hodnota má být systémem řízena.
- *Vstupní proměnná* - proměnná procesu, která nese hodnotu vstupů procesu
- *Manipulovaná proměnná* - Procesní proměnná, jejíž hodnotu lze měnit řídicím systémem
- *Bod nastavení* - požadovaná hodnota řízené proměnné

- *Systém s otevřenou smyčkou* - systém, v němž se informace o proměnných procesu nepoužívají pro nastavení systému
- *Systém s uzavřenou smyčkou* - systém, v němž se informace o procesních proměnných používá k ovládání procesní proměnné pro vyrovnání odchylek procesních proměnných od požadovaných hodnot
- *Řídicí systém se zpětnou vazbou* - řízená proměnná se sleduje za účelem ovládání jedné nebo několika procesních proměnných
- *Řídicí systém s přímou vazbou* - měří se některé procesní proměnné a očekávané poruchy se kompenzují dříve, než se projeví měřením řízené proměnné Účelem řídicího systému je udržovat specifikované vlastnosti výstup; systému na dostatečně blízkých stanoveným požadovaným hodnotám. Je-li vstupní informace konstantní, proces je plně determinován a operace jsou opakovatelné, může proces probíhat bez jakékoli kontroly. Takovému systému se říká "otevřená smyčka" a je to, jako když na sporáku nastavíme hořák na určitou intenzitu, a dále ohříváme mez kontroly průběhu ohřevu, teploty atd. V reálném světě jsou mnohem častější systémy s uzavřenou smyčkou, v níž se hodnota výstupní veličiny monitoruje a k dosažení její požadované hodnoty se ovládá vstupní signál. Příkladem je např. vytápění s termostatem.

Existují dvě obecné varianty řízení uzavřenou smyčkou. Zpětnovazební (feedback) řízení ovládá proces na základě řízení výstupní řízené proměnné. Informace o hodnotě řízené proměnné se vrací před řídicí proces a na základě porovnání s požadovanou hodnotou řízené proměnné koriguje vstupní signály k dosažení dostatečně malé odchylky řízené proměnné od požadované hodnoty.

Dopředné řízení (feedforward) ovládá proces na základě předvídání budoucího účinku změny vstupního signálu. Na základě porovnání vstupního signálu s požadovanou hodnotou vzniká signál, který ovládá proces tak, aby změna vstupu nezpůsobila odchýlení řízené proměnné od požadovaného stavu a aby si výstupní proměnná udržela požadovanou hodnotu.

2.7.2. Softwarové paradigma procesního řízení

Většinou posuzujeme programové systémy algoritmicky. Výstupy počítáme výhradně na základě zpracování vstupů. Takový normální model nepřipouští vnější poruchy a pokud se nevstupní proměnné mění spontánně, je to považováno za chybu. Takový normální softwarový model odpovídá řízení s otevřenou smyčkou. Pokud ale nejsou hodnoty všech proměnných systému zcela předvídatelné, pak se koncepce čistě algoritmického modelu hroutí. V případě, že je zpracování systému ovlivněno vnějšími poruchami, vnějšími silami nebo událostmi, které nejsou přímo viditelné nebo řiditelné algoritmicky, pak je řídicí paradigma vhodným nástrojem řešení

problému.

Architektonický styl pro software, který řídí spojité procesy, může být založen na procesně řídicím modelu, který zahrnuje tyto základní části řídicí smyčky:
A. Výpočetní prvky: oddělují doménu zájmu procesu od řídicí filozofie
- Definice procesu - zahrnuje mechanizmus pro manipulaci s určitou procesní proměnnou
- Řídicí algoritmus - rozhoduje, jak se procesní proměnnou ovládá, včetně modelu toho, jak se procesní proměnná vztahuje ke stavu procesu

B. Datové prvky: spojitě aktualizované procesní proměnné a čidla, která poskytují jejich hodnotu
- Procesní proměnné - včetně vstupních, řízených a ovládaných proměnných a informace o tom, jak se získává jejich hodnota
- Nastavené (žádané) hodnoty nebo referenční hodnoty pro řízenou proměnnou

C. Paradigma řídicí smyčky - ustavuje relace, které pak provádí řídicí algoritmus. Shromažďuje skutečné a žádané hodnoty procesu, a dolaďuje hodnoty procesních proměnných směrem k očekávanému stavu.

V procesu můžeme oddělit algoritmickou funkčnost od vlivu vnějších poruch tak, že do jednoho (softwarového) subsystému zahrneme vlastní proces, jeho definiční vztahy a procesní proměnné. V rozhraní tohoto subsystému jsou vstupní a řízené proměnné. Druhý subsystém zahrnuje řídicí algoritmus a požadované hodnoty. Tento řídicí subsystém (řadič) má spojitý přístup k nastavovacím a monitorovaným hodnotám. Pro zpětnovazební systém je jím řízená proměnná. Mezi uvedeným subsystémy jsou dvě základní interakce:
- řídicí jednotka dostává informace od procesní proměnné procesu
- řídicí jednotka dodává na vstup procesního subsystému hodnoty odchylek od požadovaných hodnot

Výsledkem je jistý druh architektury založené na toku dat. Primární charakteristikou takové architektury je , že interakce komponent závisí na datech poskytovaných jednou komponentou té druhé. Většina systémů založených na toku dat zahrnuje nezávislé (vetšinou paralelní) procesy a krokování (časovou osu), která závisí na rychlosti, se kterou si subsystémy poskytují data. Paradigma řídicí smyčky předpokládá, že data vztažená k procesní proměnné jsou aktualizována spojitě. Na rozdíl od většiny systému s tokem dat předpokládá řídicí smyčka cyklickou topologii. Řídicí smyčka ustavuje vnitřní asymetrii mezi řídicím a procesním prvkem.

2.8 Jiné známé architektury

Existuje řada dalších architektonických stylů, z nichž některé jsou obecnější, zatím co jiné jsou specifické pro jistou problémovou doménu. > Distribuované procesy: existuje řada společných způsobů organizace pro multiprocesní systémy. Některé jsou charakteristické svou topologií (prstenové uspořádání, hvězdicové uspořádání aj), jiné jsou charakteristické způsobem (protokoly) vzájemné komunikace.
- Jednou z rozšířených architektur je systém "klient-server". V těchto systémech server poskytuje služby klientům. Zatím co server nezná počet a identitu svých klientů, klient vždy zná identitu svého serveru.
- Organizace typu hlavní program/podprogram: Značné množství organizací odráží

vlastnosti programovacího jazyka, v němž je systém zapsán. Programovací jazyk, který nemá dostatečnou podporu pro modularizaci, nabízí nejčastěji organizaci v podobě hlavního programu a podprogramů. Hlavní program pak slouží jako řídicí program a zpětná vazba je realizována prostřednictvím podprogramů, vyvolaných v určité sekvenci.

- Doménově specifické architektury: V poslední době se rozvinulo úsilí, hledat pro určitou třídu problému nejvhodnější "referenční" architekturu, šitou co nejlépe na míru dané třídy problémů. Specializací architektury se usnadní vývoj deskriptivních nástrojů a struktur systému. Pak lze vyvinout také nástroje na úrovni 4GL, které dovolují vývoj automaticky nebo semi-automaticky vytvořených programových systémů.
- Stavové systémy (automaty): jsou charakteristické pro nejširší použití v různých systémech. Systém je definován množinou stavů a množinou podmínek pro přechod systému z jednoho stavu do druhého.

2.9 Heterogenní systémy

Až doposavad jsme hovořili o "čistých" architektonických stylech. Většina reálných systémů však umožňuje kombinaci několika architektonických stylů v rámci jednoho systému.

Jednou z možností "čisté" kombinace stylů je využití hierarchické struktury, která dovoluje, aby na různých úrovních struktury byly použity různé architektonické styly. Druhá možnost dovoluje aby jedna komponenta použila konektory různých architektonických stylů. Např. komponenta může s jednou komponentou komunikovat prostřednictvím konektorů stylu repozitář (sklad) a s jinou komponentou prostřednictvím roury.

Tak charakteristika jednotlivých architektonických stylů vytváří taxonomii stylů, které se mohou použít v inovativním způsobem při tvorbě programového systému.

3. Závěr

Studium a výzkum stavebních stylů rozsáhlých programových systémů – nebo architektury rozsáhlých programů, jak zní již zavedený a vznešený název převzatý z oblasti s mnohatisíciletou tradicí – vede k další a vyšší rovině systemizace programátorské práce. Taxonomie odděluje charakteristické rysy jednotlivých stylů, ale také vyčleňuje jejich společné rysy. Vytváří podmínky pro budování specializovaných podpůrných prostředků a systémů. Vytváří také pravidla a doporučení formující vyšší kulturu tvorby rozsáhlých programových děl, jejímž důsledkem by měla být nejen ekonomická efektivnost, ale také společenská přijatelnost a profesní účelnost prostředků vytvořených s respektováním nalezených zákonitostí. Architektura programových systémů se tak stává novou dimenzí disciplíny ať už ji nazýváme softwarovým inženýrstvím, nebo technologií programování.

4. Literatura

1. SHAW Mary , GARLAN David: SOFTWARE ARCHITECTURE, Perspectives On An Emerging Discipline. Carnegie Melon University, 1996, Prentice Hall, Upper Saddle River, New Jersey 07458 ISBN 0-13-182957-2.
2. Gamma,E., Helm,R., Johnson,R.,Vlissides,J.: Fesign Petterns, Elements of Reusable Object-Oriented Software. Addison Wesley Publ.comp. 1995. ISBN 0-201-63361-2.
3. Booch,G.: Object Oriented Analysis abd Design with Applications. Addison Wesley Publ. Comp. 1994. ISBN 0-8053-5340-2.

Prof. Ing. Jan M Honzík,CSc., Ústav Informatiky a výpočetní techniky FEI VUT v Brně,
http://www.fee.vutbr.cz, honzik@dcse.fee.vutbr.cz