Česká zemědělská univerzita v Praze
Provozně ekonomická fakulta

Distribuované objektové architektury

Disertační práce z oboru Informační management

Ing. Jiří Brožek

Školitel: doc. Ing. Vojtěch Merunka, Ph.D.

Chtěl bych na tomto místě poděkovat svému vedoucímu, doc. Ing. Vojtěchu Merunkovi, Ph.D., za veškeré rady a péči, kterou mi jako svému vůbec prvnímu doktorandovi věnoval.

Velké díky patří mým rodičům a hlavně pak mým milovaným děvčatům – manželce Lence, která mi je velkou a hlavně trpělivou oporou, a dcerce Kristýnce, která svým úsměvem dokáže projasnit i ten nejošklivější den.

Jiří Brožek

Hostivice, leden 2012

Distribuované objektové architektury

Klíčová slova

objekt, služba, modelování, formální model, návrhový vzor, servisně orientovaná architektura, SOMF

Abstrakt

Práce se zabývá možností využití technik z oblasti objektového modelování v oblasti servisně orientované architektury. Cílem práce je najít takovou shodu mezi objektovým a servisním modelem, která by umožnila aplikaci vybrané techniky objektového modelování při návrhu servisně orientovaného systému, a navrhnout způsob jakým tuto techniku využít.

Distributed Object Architectures

Keyword

object, service, modeling, formal model, design pattern, service oriented architecture, SOMF

Abstract

This dissertation deals studies the posibility of using object oriented modeling techniques during if the field of service oriented systems. The aim of this work is to find such analogy between object oriented and service oriented paradigms, that would allow practitioners to use selected object modeling techniques for service oriented system design, and propose how could such technique be used.

Obsah

1 Úvod

Potřeba komunikace je odvěkým prvkem lidské společnosti. V současnosti, kdy moderní informační a komunikační technologie prorůstají stále více a více našimi běžnými životy, je využití snadných komunikačních technologií a aplikací nezbytné. Využití počítačových technologií se stává běžným prvkem našeho života.

Počítačové sítě prošly v minulosti rozsáhlým rozvojem. Původní účel, kterým bylo sdílení informací mezi jednotlivými připojenými počítači, se tak díky rozšíření osobních počítačů stal jen jednou z mnoha možností, které současné počítačové sítě nabízejí. Moderní sítě se stávají nástrojem umožňujícím kromě sdílení informací i sdílení zařízení, paměťové a výpočetní kapacity.

S rozvojem sítí je spjat i rozvoj softwarových prostředků, které jsou při práci s nimi využívány. Moderní programovací jazyky umožňují práci se síťovými prostředky a některé na síťovém prostředí dokonce staví, jako na základním kameni své existence. Masivní rozšiřování objektového paradigmatu v tvorbě softwaru tak společně s využitím síťových technologií vedlo ke zrodu distribuovaného výpočetního modelu.

Bouřlivý rozvoj Internetu v posledních 15 letech a nové hardwarové i softwarové technologie umožňují dále rozvíjet myšlenku distribuovaných systémů a stojí i u zrodu servisně orientované architektury (SOA), nového paradigmatu navazujícího na servisní rysy distribuované objektové architektury. Servisně orientovaná architektura jako taková není metodikou, ale spíše filozofií a proto na ní lze nahlížet z mnoha pohledů a k její implementaci lze přistupovat různě.

Tato práce je zaměřena především na vývoj podnikových informačních systémů a na možnost využití stávajících technik objektového modelování v oblasti návrhu servisně orientovaných systémů.

Při vývoji software se začíná uplatňovat přístup, že se procesy uvnitř podniku nemají přizpůsobovat software, ale naopak software má být šitý na míru obchodním procesům uvnitř organizace. Nezastupitelnou součástí vývoje software se tak stává komplexní analýza obchodních procesů organizace. Na těchto principech staví moderní metody vývoje, jakou je například MDA – Model Driven Architecture (OMG MDA, 2010). I servisně orientovaná architektura si jako základní kámen bere model obchodních procesů (Erl, 2006), (Štumpf, 2006).

Správně navržený a implementovaný informační systém podniku představuje v současné době jednu z hlavních konkurenčních výhod organizace v téměř každém odvětví ekonomiky.

2 Motivace

Jedním z hlavních důvodů ke zvolení tématu této práce byla skutečnost, že její autor se tématikou distribuovaných objektových systémů zabýval už v rámci své bakalářské a posléze diplomové práce. Oblast vývoje software a související technologie se však živelně rozvíjí, myšlenka samotných distribuovaných objektových systémů pomalu ustupuje do pozadí a na její místo nastupují nové technologie a přístupy, například servisně orientovaná architektura. SOA bývá označováno za pokračování vývoje distribuovaných objektových systémů (viz např. práce Douglase Barryho, který již v roce 2003 v pojednání (Barry, 2003) označoval SOA jako pokračovatele myšlenky distribuované objektové technologie CORBA). Autor práce se proto rozhodl zvolit SOA jako cílovou oblast své práce a objektové technologie, potažmo postupy z oblasti jejich modelování, využít jako prostředky pro případné zlepšení této oblasti.

Motivaci k vytvoření této práce také částečně přinesla skutečnost, že během studia literatury autor narazil na názory, že by bylo vhodné, aby SOA jako přístup k tvorbě software získala i nějaké jednotné formální pozadí, přičemž tuto myšlenku vyjadřovali i autoři se SOA spojení, např. publicista Rockford Lhotka, který se SOA dlouhodobě zabývá (Lhotka, 2009).

Dalším motivujícím prvkem byla skutečnost, že se autor práce podílí na vědecké činnosti související s modelovací metodou BORM, která vznikla na Provozně ekonomické fakultě ČZU v Praze jako jeden z výsledků výzkumu katedry informačního inženýrství (více o BORM např. v (Knott, a další, 2003)). Tato metoda je postavena na objektovém základu, využívá transformace mezi modely a obsahuje již část prostředků využitelných při návrhu servisně orientovaných systémů (např. sofistikovaný aparát pro modelování procesů a validaci v něm vytvořených procesních modelů s využitím simulace). Výsledky získané při tvorbě této práce by tak mohly

být v budoucnu dále využity při dalším rozvoji této metody, například v podobě jejího rozšíření přímo určeného pro modelování SOA.

3 Cíle

Tato práce je zaměřena na problematiku návrhu distribuovaných informačních systémů se zaměřením na servisně orientovanou architekturu.

Hlavním cílem práce je navrhnout možnost aplikace vybrané techniky objektového modelování při řešení problému v oblasti modelování servisně orientovaných systémů s možnou aplikací v modelech servisně orientovaného modelovacího frameworku SOMF.

Pro splnění hlavního cíle je nutné stanovit dílčí cíle:

1. analyzovat literární zdroje týkající se dané problematiky,
2. na základě této analýzy nalézt ontologicky analogické prvky objektově orientovaného a servisně orientovaného paradigmatu,
3. tyto analogické prvky popsat s využitím formálního aparátu vybraného objektového kalkulu,
4. na základě analogie navrhnout jakým způsobem aplikovat návrhové vzory z oblasti objektového modelování v oblasti modelování servisně orientovaných systémů.

4 Metodika

V první části práce bude zpracována analýza literárních pramenů zaměřených na vývoj informačních systémů. Budou zmapovány základní architektury informačních systémů a jejich vlastnosti. Zvláštní důraz bude kladen na platformy využívající objektového přístupu a servisně orientovanou architekturu, včetně modelování softwarových systémů pro implementaci pomocí objektového a servisního paradigmatu.

Na základě této analýzy budou formou ontologické analogie vyhledány společné prvky objektových a servisně orientovaných modelů, jejichž podobnost by umožnila aplikaci techniky návrhových vzorů objektového modelování v oblasti servisně orientovaného paradigmatu.

Nalezené analogické prvky budou formálně popsány pomocí objektového kalkulu definovaného v (Merunka, a další, 2007) a (Merunka, 2008) a na základě tohoto popisu budou vybrané návrhové vzory z oblasti objektového modelování formálně popsány a bude navržena možnost jejich využití v oblasti modelování servisně orientovaného paradigmatu.

5 Softwarová architektura a vývoj výpočetního modelu

5.1 Softwarová architektura

Softwarovou architekturou systému rozumíme množinu softwarových prvků, jejich vlastností a vztahů mezi nimi (Clements, 2009). Standard IEEE tuto definici zpřesňuje a uvádí následující definici:

> *„Architecture is the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution."*[1] (IEEE 1471-2000)

David Garlan a Mary Shaw, kteří se problematikou softwarových architektur zabývají již od počátku 90. let minulého století, v (Garlan, a další, 1993) poukazují na to, že tato problematika se stala aktuální s rozvojem vyšších programovacích jazyků. Zatímco v prvních programovacích jazycích datové typy sloužily víceméně k rozhodnutí pro použití správné strojové instrukce, v modernějších jazycích už programátor pomocí výběru použitých datových typů přímo říká, co s daty zamýšlí a jak by měla být použita. Tyto rozdíly ještě více podtrhují možnosti použití abstraktních datových typů, rozdělení zdrojového kódu na jednotlivé moduly a oddělení samotné implementace modulu od specifikace jeho rozhraní.

Na tyto myšlenky navazují v (Garlan, a další, 1996), kde mimo jiné představují i myšlenku architektonických stylů, která je platná dodnes:

> *[An architectural style] defines a family of systems in terms of a pattern of structural organization. More specifically, an architectural style*

[1] *Architektura je základní uspořádání systému sestávající z jeho komponent, vztahů mezi nimi, vztahů s prostředím, a principů jeho návrhu a dalšího rozvoje.*

defines a vocabulary of components and connector types, and a set of constraints on how they can be combined.[2]

Mezi známé architektonické styly patří např. (podle (Garlan, a další, 1993) (Mylopoulos, 2003)):

- Blackboard
- Peer-to-peer
- Pipes and filters
- Monolitické aplikace
- Model klient-server
- Model-view-controller
- Komponentový model
- Třívrstvá architektura
- Servisně orientovaná architektura
- …

Definice architektury podle (IEEE 1471-2000) zmiňuje termín systém ve smyslu vazby jednotlivých komponent a prostředí. Aby mohl být software využíván, musí být spuštěn. Aby mohl být spuštěn, musí být nasazen na nějakém hardware. Výsledný systém je potom kombinací software a hardware a pouze touto kombinací lze dosáhnout požadovaných vlastností (spolehlivost, výkonnost, atp.). Bez hardware nemůže samotný software těchto vlastností dosáhnout (Eeles, 2006).

Podle této definice tedy můžeme vyvozovat, že architektonické styly jsou v mnohém vázány na konkrétní hardwarové implementace, výpočetní modely i programovací jazyky. Dokazuje to například členění výpočetních modelů dle (Peterka, 2005), ve kterém se shodně vyskytují model *klient-server* i *třívrstvá architektura*. Podobně styl *model-view-controller*, který

[2] *Architektonický styl definuje rodinu systémů v podobě vzoru stavebního uspořádání. Přesněji, architektonický styl definuje slovník komponent a typů spojek, a množinu omezení jejich vzájemné kombinace.*

vznikl původně jako framework jazyka Smalltalk (Reenskaug, 1979), bývá řazen i do kategorie návrhových vzorů (Martin, 2002).

5.2 Vývoj výpočetního modelu

Výpočetní model nám dává představu o tom, kde jsou aplikace uchovávány a kde skutečně spouštěny, zda a případně jak jsou rozděleny na součásti, kde se uchovávají a jak se zpracovávají data a kde se nachází uživatel a jakým způsobem s aplikací a daty pracuje (Peterka, 2005).

Výpočetní model prošel za posledních 50 let bouřlivým vývojem. Počínaje vysoce centralizovanými systémy (mainframe architektura), přes terminálové zpracování, monolitické aplikace až po distribuované architektury různého druhu, včetně novinek v podobě grid či cloud computingu (Li, a další, 2010).

5.2.1 Dávkové zpracování

Jedná se o historicky nejstarší model výpočtu. Byl vynucen vysokými náklady na využití výpočetní techniky, málo výkonným hardware a software. Jedná se o model výpočtu, který není interaktivní. Uživatel si připravil úlohu ke zpracování (dávku), která byla zařazena do fronty. Úlohy zařazené do fronty byly zpracovávány sekvenčně jedna po druhé a jejich výstup byl zpracován do určité formy, například textové výstupní sestavy. Dávka zadávaná uživatelem krok po kroku popisovala algoritmus zpracovávané úlohy (např. výpočtu). (Peterka, 2005)

5.2.2 Model host/terminál

Je reakcí na neinteraktivnost dávkového zpracování. Umožňuje současnou práci více uživatelů a interaktivní styk uživatele s úlohou. Na rozdíl od dávkového zpracování umožňuje na straně hosta zpracování více úloh najednou. Zprvu pomocí metody time-sharing (McCarthy, 1996), posléze i pomocí preemptivního multi-taskingu.

5.2.3 Izolované počítače

Příchod osobních počítačů na počátku 80. let 20. století způsobil přeliv od absolutní centralizace (mainframy) k absolutní decentralizaci v podobě mnoha počítačů, které spolu nebyly nijak propojené. Výhodou byla skutečnost, že každý uživatel měl svůj počítač, čímž se zvýšil uživatelský komfort. Nevýhodami bylo například to, že vyvstaly problémy se sdílením dat nebo správou aplikací. Finančně nákladné periferie také nebylo možné pořídit ke každému počítači v organizaci. (Peterka, 2005)

5.2.4 Model file-server/work-station

Model file-server/work-station vznikl jako řešení některých problémů, které se vyskytly při použití izolovaných počítačů. Přináší centralizované body (file-servery), které umožňují sdílení dat. Data, která je potřeba sdílet tak mohou být sdílena, ale zároveň je zachována individuálnost a izolovanost toho, co sdíleno být nemusí. Sdílena mohou být jak data, tak samotné aplikace, které jsou sice fyzicky uloženy na vzdáleném úložišti souborového serveru, ale spouštěny jsou přímo na dané pracovní stanici. (Peterka, 2005)

5.2.5 Model klient/server

Při využití modelu file-server/work-station bylo nutné přenášet velké objemy dat i v případě, že byl využit jen zlomek z nich. Modelu klient/server vznikl aby tento problém odstranil. Jeho základní myšlenkou je rozdělení dosavadní monolitické aplikace na dvě části. Ta část aplikační logiky, která pracuje s daty, je spolu s nimi uložena na vzdáleném serveru a všechny potřebné operace se tak provádějí přímo na jednom místě. Druhá část aplikace, prezentační vrstva, je umístěna na uživatelském počítači. Pokud je aplikace vhodně navržena, je nutný datový přenos mezi centrálním serverem a uživatelským počítačem minimální. (Peterka, 2005)

Aplikace na straně uživatelského počítače je klientem, kterého obsluhuje vzdálený služebník – server. Protože klientská aplikace obsahuje stále velkou část aplikační logiky, je nazývána *tlustý klient* (fat klient).

5.2.6 Třívrstvá architektura

Tento model je rozšířením modelu klient/server. Pro mnohé aplikace přestávalo být rozdělení na dvě části dostačujícím. Třívrstvá architektura provádí další rozdělení aplikace a přidává další vrstvu. Aplikace je tedy rozdělena na tři části s nezastupitelnou funkcí:

- správa dat a práce s nimi – obvykle databázový server
- aplikační logika – obvykle aplikační server
- prezentační část – uživatelské rozhraní sloužící k obsluze aplikace, sběru dotazů a prezentaci výsledků

V současné době velmi oblíbený model, k prezentaci často využívána forma webového sídla a na straně uživatelského počítače je vyžadován jen webový prohlížeč (Peterka, 2005). V souvislosti s třívrstvou architekturou se velmi často objevuje i termín *tenký klient* (thin klient). V případě webového sídla jako prezentační vrstvy však lze o tomto termínu polemizovat, protože současný web velmi často využívá klient-side technologie typu Ajax, Flash či Silverlight, které sice v lecčem ulehčují serveru, ale přenáší tak část aplikační logiky na klientský počítač.

5.2.7 Distribuovaná objektová architektura

Vznikla jako jedna z reakcí na nedostatky architektury klient/server. První aplikace se datují do počátku 90. let 20. století. Myšlenkou distribuované architektury je rozložit aplikaci na několik funkčních modulů, které mohou fyzicky existovat na různých počítačích v rámci počítačové sítě nebo v rámci různých procesů na jednom počítači. Objekty mezi sebou komunikují pomocí zasílání zpráv a příjmu odpovědí, lhostejno, zda se jedná o objekty lokální nebo vzdálené. (Emmerich, 2000), (Caromel, a další, 2005)

Nejrozšířenější distribuované objektové architektury budou podrobněji popsány v kapitole 6.

5.2.8 Servisně orientovaná architektura

Servisně orientovaná architektura je jedním z možných způsobů návrhu a integrace softwarových systémů. V určitých rysech je myšlenkovým pokračovatelem distribuovaných technologií jako je CORBA (Barry, 2003). Není však technologií, ale spíše návrhovou filozofií (Erl, 2006). Jejím základním kamenem jsou služby, které uživatelé mohou využívat ve svých aplikacích. Je lhostejné, v jakém programovacím jazyce jsou služby implementovány. Důležitým atributem SOA je volná vazba mezi jednotlivými službami. Tato architektura přímo podporuje a povzbuzuje znovu použitelnost vytvořených služeb. (Erl, 2007)

O servisně orientované architektuře blíže pojednává kapitola 7.

6 Distribuované objekty v tvorbě aplikací

Objektově orientované jazyky, jako je *C++*, *C#* nebo *Java*, poskytují oproti standardním procedurálním jazykům značné výhody, co se týče opětovné použitelnosti a modularity zdrojového kódu.

Deborah Armstrong v (Armstrong, 2006) uvádí tzv. kvarky objektového vývoje – vlastnosti, které jako klíčové označuje většina autorů textů o objektovém programování, a jejich definice[3]. Nejčetnějšími termíny jsou:

- dědičnost,
- objekt,
- třída,
- zapouzdření,
- metoda,
- předávání zpráv,
- polymorfismus,
- abstrakce.

Na základě (Armstrong, 2006) a (Merunka, 2008) můžeme popsat následující vlastnosti, které mají klíčový význam pro objektové programování:

- *Třída* v objektově orientovaném paradigmatu označuje aparát umožňující vytvořit společný popis množiny dat a množiny metod daného druhu objektů.
- *Dědičnost* umožňuje rozšíření a modifikaci knihovny kódu a dat bez nutnosti vlastnit a přímo upravovat zdrojový kód původní knihovny.
- *Zapouzdření* nabízí možnost tvorby knihoven rozhraní, které minimalizují závislost na algoritmech a datových strukturách, které se za nimi skrývají. V těchto vnitřních strukturách pak

[3] Autorka provedla rozbor textů o objektově orientovaném programování z let 1966-2005 a zjišťovala, které jejich autoři označují za základní vlastnosti objektového vývoje.

mohou být později prováděny změny, aniž by bylo nutné modifikovat kód, který tyto knihovny používá.

- *Polymorfismus* umožňuje části kódu pracovat s několika různými datovými typy, aniž by se kvůli jejich různosti musel měnit.

Z pohledu distribuované objektové technologie rozlišujeme jazyky „standardní“ a jazyky s přímou podporou distribuované objektové technologie.

Zatímco "standardní" objektově orientované programovací jazyky dovolují budovat monolitické programy složené z mnoha objektů – stavebních bloků, distribuované objektové technologie, jako je CORBA sdružení OMG, DCOM a .NET společnosti Microsoft či Java RMI společnosti Oracle, umožňují tvorbu software, jehož komponenty jsou volně rozprostřeny po několika počítačích v síti. (Emmerich, 2000)

Při implementaci vztahu klient-server mezi jednotlivými objekty používají distribuované objektové systémy registrový mechanismus (registr v případě CORBA se nazývá Object Request Broker, nebo-li ORB), který slouží k ukládání a sdílení popisu rozhraní dostupných objektů. Pomocí služeb ORB může klient transparentně volat metodu objektu na vzdáleném serveru. ORB je zodpovědný za nalezení objektu, který může implementovat požadavek, předání parametrů, zavolání metody a návrat výsledku. Klient si vůbec nemusí být vědom, kde se daný objekt nachází, v jakém programovacím jazyce je implementován, na jakém operačním systému běží, ani dalších systémových aspektů, které nejsou součástí rozhraní objektu. (Caromel, a další, 2005)

Následující kapitoly popisují nejpoužívanější technologie pro práci s distribuovanými objekty v jazyce Java.

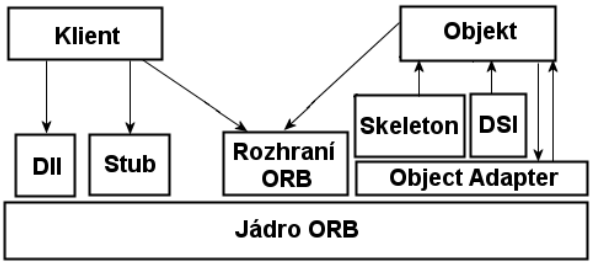
6.1 Architektura CORBA

Common Object Request Broker Architecture (CORBA) je otevřenou architekturou pro distribuované objektové výpočty, kterou standardizuje

Object Management Group (OMG). CORBA automatizuje mnoho obvyklých programátorských úloh v síťových aplikacích, jako je například registrace, nalezení a aktivace objektů, demultiplexování požadavků, řešení chybových stavů, předávání požadavků atp.

6.1.1 Architektura CORBA ORB

Následující obrázek ilustruje komponenty architektury CORBA ORB. Popis jednotlivých komponent podle specifikace (Object Management Group, 2008) je uveden níže.



Obr. 6.1. Architektura CORBA ORB (vlastní zpracování autora podle OMG)

- **Objekt** – entita mající identitu, rozhraní a implementaci, která se nazývá *Servant*.
- **Servant** (služebník) – implementace entity v programovacím jazyce, která definuje operace pokryté pomocí rozhraní CORBA IDL. Servant může být napsán v mnoha programovacích jazycích, včetně C, C++, Java, Smalltalk a Ada.
- **Klient** – Programová entita volající metodu implementace objektu. Zpřístupnění služeb vzdáleného objektu by mělo být pro volajícího transparentní. V ideálním případě by mělo být stejně snadné jako normální volání metody, tj. `objekt.metoda(parametr)`. Zbývající komponenty na obrázku umožňují tuto úroveň transparence.
- **Object Request Broker (ORB)** – ORB poskytuje mechanismus pro transparentní předání klientského požadavku cílovému vzdálenému objektu. ORB zjednodušuje distribuované programování tím, že od klienta odstíní všechny detaily vzdáleného volání metody. Pro klienta

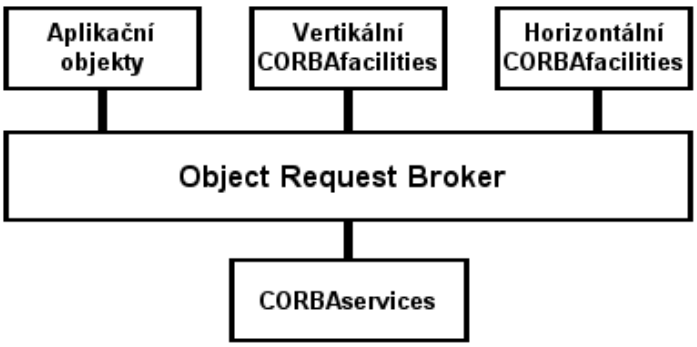
pak všechny požadavky vypadají jako volání místní metody. Když klient volá metodu, je odpovědností ORB nalézt implementaci objektu, pokud je to nezbytné ho aktivovat, doručit objektu požadavek a vrátit jeho odpověd volajícímu.

- **ORB Interface** – ORB je logická entita, která může být implementována mnoha způsoby (například jeden či více procesů nebo sada knihoven). Aby byla aplikace oddělena od implementačních detailů, definuje specifikace CORBA pro ORB abstraktní rozhraní. Toto rozhraní obsahuje mnoho pomocných funkcí, jako např. konverzi odkazů na objekty na řetězce znaku a obráceně, a vytváření seznamu argumentů pro požadavky provedené pomocí dynamického rozhraní popsaného níže.
- **CORBA IDL stuby a skeletony** – Stuby a skeletony CORBA IDL slouží jako pojítko mezi klientem a serverem a ORB. Transformace mezi definicemi v CORBA IDL a cílovým programovacím jazykem je automaticky prováděna kompilátorem CORBA IDL. Použití kompilátoru snižuje možnost případné nekonzistence mezi klientskými stuby a serverovými skeletony a zároveň umožňuje větší možnost optimalizace při kompilaci.
- **Dynamic Invocation Interface (DII)** – dynamické rozhraní pro volání. Umožňuje klientu přímý přístup k volacím mechanismům poskytovaným ORB. Aplikace využívají DII k dynamickému volání metod vzdálených objektů bez nutnosti použití stubů rozhraní IDL. Na rozdíl od použití stubů umožňuje DII i neblokující synchronní (samostatné operace pro odeslání a přijetí) a jednosměrná (pouze odeslání) volání.
- **Dynamic Skeleton Interface (DSI)** – Analogie k DII na straně serveru. DSI umožňuje ORB doručit požadavky implementaci objektu, která v době překladu neví jaký typ objektu implementuje. Klient vysílající požadavek neví, zda implementace používá typově specifické IDL skeletony nebo dynamické skeletony.

- **Object Adapter** – pomáhá ORB s předáváním zpráv objektům a jejich aktivací. Důležitější však je, že adaptér asociuje implementaci objektu s ORB. Objektové adaptéry mohou být specializované na určitý druh implementačního stylu.

6.1.2 Object Management Architecture

Obrázek 6.2 zobrazuje základní schéma Object Management Architecture (OMA) vytvářené OMG (Object Management Group, 2010). OMA staví na principech CORBA a vychází z předpokladu, že aplikace v určité provozní oblasti sdílejí více či méně stejnou funkcionalitu.



Obr. 6.2. Object Management Architecture
(vlastní zpracování autora podle OMG)

OMA nabízí množinu objektů, které vykonávají jasně definované funkce a jsou přístupné pomocí OMG IDL. OMG standardizuje IDL rozhraní a upřesňuje, co jednotlivé objekty dělají. Dodavatelé software pak vytváří a prodávají implementace těchto objektů, které poskytují dané služby.

Object Management Architecture se skládá z následujících 4 složek:

- **CORBAservices** – poskytují základní funkčnost. Pro distribuované aplikace poskytují služby analogické například volání API Windows či systémovým voláním v UNIXu.
- **Horizontální CORBAfacilities** – existují mezi CORBAservices a aplikačními objekty (viz níže). Existují pouze čtyři CORBAfacilities: *Printing Facility*, *Secure Time Facility*, *Internationalization Facility* a *Mobile Agent Facility*.

- **Domain (vertikální) CORBAfacilities** – neustále se rozrůstající kategorie v rámci OMA. IDL je výborný způsob standardizace rozhraní pro standardní objekty, které mohou sdílet společnosti v rámci odvětví. OMG byla oslovena několika společnostmi s žádostí o pomoc, proto byla v roce 1996 založena Domain Technology Committee, jejímž úkolem je koordinovat práci na Domain CORBAfacilities.
- **Aplikační objekty** – nejvyšší partie OMA hierarchie. Vzhledem k tomu, že tato kategorie je specifická pro každou aplikaci a nevyžaduje standardizaci, nevztahují se na ně standardizační snahy OMG.

CORBA je některými autory, např. Douglasem Barrym (Barry, 2003) uváděna jako jedna z předchozích servisně orientovaných technologií.

6.2 Nativní distribuované objektové aplikace v jazyce *Java*

Jazyk Java přímo podporuje možnosti tvorby distribuovaných objektových aplikací pomocí mechanismu Java RMI (Oracle, 2010). RMI (Remote Method Iinvocation, volání vzdálené metody) je specifickou vlastností programovacího jazyka Java společnosti Sun a je tedy vázáno pouze na použití v tomto programovacím jazyce.

Do verze Java2 1.1 RMI také bylo jedinou možností práce s distribuovanými objekty. Od verze Java2 1.2 pak jazyk Java umožňuje i využití Java IDL, které tak javovým programům otevřelo přístup k architektuře CORBA.

RMI aplikace se často skládají ze dvou částí – serveru a klienta. Typická serverová aplikace vytvoří několik vzdálených objektů, zpřístupní odkazy na tyto objekty a čeká na klienta, který bude volat metodu některého z těchto objektů. Typická klientská aplikace získá odkaz na jeden nebo více vzdálených objektů na serveru a volá jejich metody. RMI poskytuje mechanismus, který umožňuje vzájemnou komunikaci klienta a serveru a

předávání dat mezi nimi. Taková aplikace je často nazývána jako distribuovaná objektová aplikace.

Distribuovaná objektová aplikace potřebuje:

Nalézt vzdálené objekty

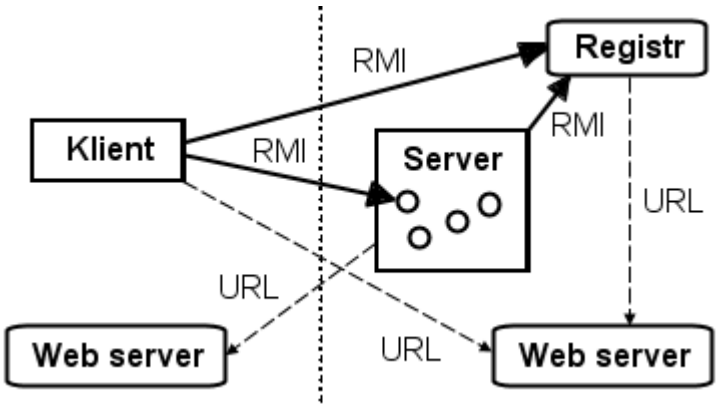
Aplikace může použít jeden ze dvou způsobů jak nalézt vzdálený objekt. Aplikace buď může vzdálené objekty zaregistrovat v jednoduchém jmenném zařízení *RMI*, tzv. `rmiregistry`, nebo může aplikace předávat a vracet odkazy na vzdálené objekty jako součást normálních operací.

Komunikovat se vzdálenými objekty

O detaily komunikace mezi objekty se starají mechanismy RMI. Z pohledu programátora je práce se vzdálenými objekty totožná jako volání místních metod.

Protože RMI umožňuje volajícímu předávat objekty vzdáleným objektům, poskytuje nezbytné mechanismy pro načtení kódu objektu stejně jako odeslání jeho dat.

Obrázek 6.3 zobrazuje RMI aplikaci využívající registr k získání informací o vzdálených objektech. Server zavolá registr, aby asocioval jméno se vzdáleným objektem. Klient si podle jména najde objekt v registru serveru a volá jeho metodu. Obrázek také ukazuje, že pokud je to nezbytné, používá RMI systém existující webový server pro načítání *bytekódu* tříd vytvořených v jazyce Java ze serveru klientovi a od klienta na server. RMI může načítat *bytekód* pomocí jakéhokoliv URL protokolu, který je platformou Java podporován (např. HTTP, FTP, soubor, atp.). (Oracle, 2010)



Obr. 6.3. Schéma RMI architektury
(vlastní zpracování autora podle (Oracle, 2010))

6.2.1 Vzdálené objekty

V distribuovaném objektovém modelu jazyka Java vzdálenými objekty rozumíme ty, jejichž metody mohou být volány z jiného virtuálního stroje (Java Virtual Machine), případně z jiného počítače. Objekt tohoto typu je popsán jedním nebo více vzdálenými rozhraními, což jsou rozhraní napsaná v jazyce Java, která deklarují metody vzdálených objektů.

Volání vzdálených metod (RMI) je akce volání metody vzdáleného rozhraní vzdáleného objektu. Nejdůležitější je, že volání metody vzdáleného objektu má stejnou syntaxi jako volání metody objektu místního.

6.3 Porovnání distribuovaných a nedistribuovaných modelů

Distribuovaný model u jazyka Java je podobný objektovému modelu v následujících ohledech (Oracle, 2010):

- Odkaz na vzdálený objekt může být předán jako parametr nebo vrácen jako výsledek při jakémkoliv volání metody (místním i vzdáleném).
- Vzdálený objekt může být přetypován na jakékoliv vzdálené rozhraní podporované implementací pomocí syntaxe pro přetypování, která je součástí jazyka Java.
- Vnitřní operátor `instanceof` může být použit k ověření vzdálených rozhraní podporovaných vzdáleným objektem.

Distribuovaný model u jazyka Java se od objektového modelu liší v těchto ohledech:

- Klienti vzdálených objektů spolupracují se vzdálenými rozhraními, nikdy však s implementačními třídami těchto rozhraní.
- Místní parametry a výsledky ze vzdáleného volání jsou předávány kopií místo odkazem. Je tomu tak proto, že odkazy na objekt jsou použitelné pouze v rámci jednoho virtuálního stroje.
- Vzdálený objekt je předáván odkazem, nikoliv kopií aktuálního vzdáleného rozhraní.
- Sémantika některých metod definovaných třídou `java.lang.Object` je pro vzdálené objekty specializována.
- Vzhledem k tomu, že jsou chybové módy při vzdáleném volání o hodně komplikovanější, musí se klient postarat o další výjimky, které mohou nastat během volání vzdálené metody.

7 Servisně orientovaná architektura

Servisně orientovanou architekturou (SOA) bývá označována množina pružných principů návrhu používaných v průběhu vývoje a integrace informačních systémů. Monografie (Erl, 2006) a (Erl, 2007) patří k nejobsáhlejším a nejucelenějším informačním zdrojům o SOA. Proto budou v této kapitole citovány jako hlavní prameny.

Erl v (Erl, 2006) uvádí následující definici SOA:

> *SOA is a form of technology architecture that adheres to the principles of service-orientation. When realized through the Web services technology platform, SOA establishes the potential to support and promote these principles throughout the business process and automation domains of an enterprise.*[4]

V (Erl, 2007) ji následně doplňuje a rozšiřuje:

> *SOA establishes an architectural model that aids to enhance the efficiency, agility, and productivity of an enterprise by positioning services as the primary means through which solution logic is represented in support of the realization of strategic goals associated with service-oriented computing.*[5]

Barry (Barry, 2003) poskytuje trochu volnější definici SOA:

> *A service-oriented architecture is essentially a collection of services. These services communicate with each other. The communication*

---

[4] *SOA je formou technologické architektury, která vyhovuje principům servisní orientace. Pokud je realizována pomocí platformy webových služeb, má potenciál podporovat a prosazovat tyto principy v oblastech obchodních procesů a automatizovaných činností v podniku.*

[5] *SOA představuje architektonický model, který pomáhá zvýšit efektivitu, agilnost a produktivitu podniku tím, že využívá služby jako primární prostředek, s jehož pomocí je prezentována logika řešení, mající za cíl podpořit uskutečnění strategických cílů spojených se servisně orientovanou výpočetní platformou.*

can involve either simple data passing or it could involve two or more services coordinating some activity.[6]

Podobně jako objektová orientace, i servisní orientace se už v současnosti stala typickou metodou návrhu. Aplikace principů servisní orientace na procesní logiku ústí ve standardizovanou servisně orientovanou procesní logiku. Pokud je aplikace složena z jednotek se servisně orientovanou procesní logikou, stává se servisně orientovanou aplikací.

Klíčové principy servisní orientace jsou následující (Erl, 2006):

- *Volná vazba* - služby udržují vztahy, které minimalizují závislosti a vyžadují pouze to, že si o jsou vědomy ostatních služeb.
- *Služební kontrakt (service contract)* - služby dodržují dohodnuté způsoby komunikace, které byly definovány hromadně v jednom nebo více popisech služeb a souvisejících dokumentech.
- *Autonomie* - služby mají kontrolu nad logikou, kterou zapouzdřují.
- *Abstrakce* - krom toho, co je popsáno ve služební smlouvě, je veškerá logika uvnitř služby skryta před vnějším světem.
- *Opětovná použitelnost* - logika je rozdělena do služeb, s cílem podporovat jejich opětovné využití.
- *Skládatelnost* - skupiny služeb mohou být koordinovány a spojeny tak, aby vytvořily složenou službu.
- *Nestavovost* - služby minimalizují uchovávání informací specifických pro aktivitu.

---

[6] *Servisně orientovaná architektura je vlastně kolekcí služeb. Tyto služby spolu komunikují. Komunikace může být buď prosté předávání dat, nebo se může jednat o koordinaci nějaké činnosti mezi dvěma či více službami.*

- *Objevitelnost* - služby jsou navrženy tak, aby byly navenek popisné, takže mohou být nalezeny a zhodnoceny pomocí dostupných vyhledávacích mechanismů.

Jednotlivé principy servisní orientace jsou podrobně rozebrány v (Erl, 2006).

SOA je obvykle spojována s webovými službami, nicméně není pravda, že SOA může využívat pouze webové služby (Štumpf, 2006) (Erl, 2006). Služby mohou být implementovány s využitím mnohých technologií a programovacích jazyků. Lze využít i technologií, které jsou považovány za přímé předchůdce dnešní podoby SOA, např. tedy technologie CORBA nebo DCOM (Barry, 2003).

Orientace na služby vyžaduje volnou vazbu služeb na operační systémy a další technologie, na kterých jsou aplikace postaveny. SOA odděluje funkcionalitu do samostatných jednotek nebo služeb (Bell, 2008) (Erl, 2007), které jejich tvůrci zpřístupňují pomocí počítačové sítě a umožňují tak uživatelům jejich kombinaci a znovupoužití při tvorbě aplikací. Tyto služby a jejich korespondující uživatelé spolu komunikují předáváním dat v dobře definovaném sdíleném formátu, nebo koordinací aktivit mezi dvěma nebo více službami (Bell, 2010).

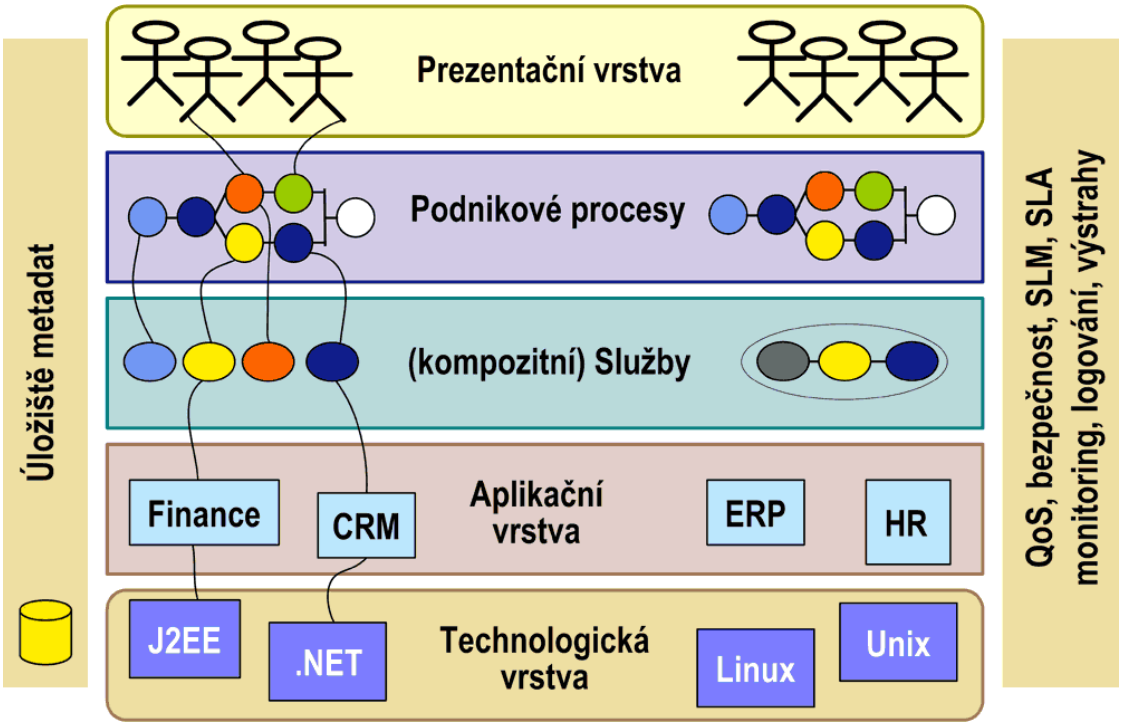
Implementace SOA závisí na síti softwarových služeb. Služby zahrnují neasociované, volně spojené funkční jednotky, které přímo v sobě nemají zahrnuty žádná vzájemná volání. Každá služba implementuje jednu činnost, například registrační formulář nebo výpis transakcí bankovního účtu. Místo toho, aby služba přímo v sobě obsahovala volání jiné služby, využívá se definovaných protokolů, které s využitím meta dat popisují, jak služby předávají a interpretují zprávy. (Erl, 2006) (Erl, 2007)

Jednotlivé SOA objekty jsou spojovány pomocí tzv. orchestrace. V průběhu procesu orchestrace tvůrce spojuje patřičné služby do podoby ne-hierarchické struktury. Využívá k tomu softwarových nástrojů, které obvykle obsahují seznam dostupných služeb, jejich popis a prostředky k vybudování aplikace využívající tyto zdroje. (Erl, 2006).

Cílem je, aby uživatel mohl v ideálním případě sestavit aplikaci s využitím pouze existujících služeb. Reálně však bude těžké ideálního stavu dosáhnout, protože mnohé aplikace jsou příliš doménově omezené. (Štumpf, 2006)

7.1 Referenční model SOA

Referenční model je abstraktní rámec pomáhající porozumět vztahům mezi entitami v určitém prostředí. Dále pak slouží softwarovým architektům pro návrh konkrétní architektury software.



Obr. 7.1. Rozšířený referenční model SOA podle Štumpfa (Štumpf, 2006)

Referenční modely prezentované jednotlivými dodavateli software se mírně liší a je velmi těžké najít model, který by byl implementačně nezávislý. Štrumpf v (Štumpf, 2006) představuje referenční model založený na modelu firmy CBDI, který rozšiřuje o další vrstvy.

Jak vyplývá ze schématu referenčního modelu na obrázku 7.1, nutným předpokladem pro vytvoření správně fungujícího servisně orientovaného systému jsou dobře zmapované obchodní procesy v rámci podniku. Na

základě tohoto mapování jsou vytvářeny a orchestrovány konkrétní služby. Jak dále ze schématu vyplývá, jednotlivé služby mohou mít různou složitost, tvořit složené služby a hlavně, jak už bylo řečeno dříve, mohou být implementovány rozličnými technologiemi. Vybrané prostředky pro mapování procesů budou popsány v kapitole 8.

7.2 Manifest SOA

Po vzoru Agilního manifestu vznikl v roce 2009 Manifest SOA, pod kterým je podepsáno 17 významných osobností z této oblasti. Tito lidé prostřednictvím manifestu říkají, že dávají při využití SOA přednost:

- **obchodní hodnotě** před technickou strategií,
- **strategickým cílům** před ziskem z jednoho projektu,
- **přirozené součinnosti** před zakázkovou integrací,
- **sdíleným službám** před jednoúčelovými implementacemi,
- **pružnosti** před optimalizací,
- **zpřesňování vývojem** před honbou za dokonalostí hned v počátku.
  (SOA Manifesto, 2009)

Při návrhu nových SOA řešení podle filozofie manifestu by tak měl být brán ohled na tato výše zmíněná kritéria.

8 Vybrané prostředky a techniky pro modelování objektově a servisně orientovaných systémů

8.1 Objektové modelování a návrh

8.1.1 Unified Modeling Language

Unified Modeling Language (UML, unifikovaný modelovací jazyk) je standardizovaný víceúčelový modelovací jazyk, primárně cílený na oblast objektového návrhu a vývoje software. UML je spravováno a dále rozvíjeno sdružením Object Management Group. V současné době je UML průmyslovým standardem pro modelování softwarových systémů.

UML je nejvíce využívanou specifikací z portfolia OMG, která je využitelná nejen pro modelování struktury aplikace, jejího chování, ale také procesů a datových struktur. Je také jedním ze základů Model Driven Architecture, která sjednocuje všechny kroky vývoje a systémové integrace, počínaje procesním modelováním, přes architektonické a aplikační modely, po vývoj, nasazení, údržbu a rozvoj.

UML obsahuje množinu grafických notací pro vytváření modelů objektově orientovaných systémů. Využívá se ke tvorbě specifikací, vizualizacím, modifikaci, konstrukci a dokumentování vyvíjeného softwarového systému.

Kombinuje techniky z oblasti datového modelování (entity relation diagramy), procesního modelování (work-flow), objektového a komponentového modelování. Jeho využití v průběhu vývojového životního cyklu není závislé na využívané implementační platformě.

UML je vytvářeno jako rozšiřitelné, přičemž rozšiřování je možné dvěma způsoby – pomocí stereotypů a profilů.

Stereotypy umožňují rozšiřovat slovník UML o další elementy. Ty jsou založené na elementech stávajících, ale mají specifické vlastnosti, které jsou vhodné pro patřičnou problémovou doménu.

Profily přizpůsobují UML pro využití ve specifických oblastech. Standardizované profily jsou dostupné v katalogu OMG. Ten obsahuje například profily pro Enterprise Application Integration (EAI), CORBA, CORBA Component Model, testování, systémové inženýrství a další.

8.1.2 Návrhové vzory

Návrhové vzory v softwarovém inženýrství představují obecná vícenásobně využitelná řešení problémů objevujících se při návrhu software. Návrhový vzor není zdrojový kód ani softwarový modul. Jedná se o popis řešení problému. Návrhové vzory z oblasti objektového modelování a programování obvykle znázorňují vztahy a interakce mezi třídami nebo objekty, bez toho aby specifikovaly, jaké konkrétní třídy a objekty jsou při implementaci využity.

První knižní publikace shrnující problematiku návrhových vzorů, klasifikující vzory do skupiny a nabízející jejich ucelenou knihovnu je (Gamma, a další, 1994). Ačkoliv je tato kniha takřka 20 let stará, je stále považována za „knihu knih“ a většina autorů zpracovávajících toto téma stále využívá názvů a členění vzorů definovaných v této knize (např. (Shalloway, a další, 2001), (Larman, 2005), (Pecinovský, 2007)).

8.1.2.1 Dokumentace návrhových vzorů

Každý návrhový vzor je zdokumentován. Jedním z obvyklých formátů dokumentace návrhových vzorů je formát využívaný právě v (Gamma, a další, 1994). Popis obsahuje následující části:

- Název – návrhový vzor by měl mít popisný název, který vystihuje jeho účel.

- Problém – popis konkrétní problémové situace, kterou využití návrhového vzoru může vyřešit.
- Alternativní název – další název pro vzor (např. snadněji zapamatovatelný).
- Podmínky – úspěšná aplikace vzoru může být ovlivněna dalšími okolnostmi, které by měly být popsány v této části.
- Řešení - popis jak dosáhnout požadovaného výsledku.
- Příklady – každý popis návrhového vzoru by měl obsahovat i ukázky jeho použití. Příklad by měl obsahovat definici konkrétního problému, vstupující podmínky, popis jak je návrhový vzor implementován a výsledek.
- Výsledek - stav nebo konfigurace systému po aplikaci vzoru.
- Odůvodnění a souvislosti - vysvětlení, proč byl návrhový vzor použit a jak jeho implementace vyřešila danou situaci.
- Související vzory - mnohdy jsou vstupní podmínky pro použití popisovaného vzoru výsledkem aplikace předchozího a na druhé straně výstupní kontext představuje vstupní bod pro jiný návrhový vzor. Návrhový vzor může být také založen na jiném vzoru, případně jiný vzor využívat.

8.1.2.2 Klasifikace návrhových vzorů

Základní rozdělení návrhových vzorů vychází z (Gamma, a další, 1994). Autoři vymezili základní skupiny návrhových vzorů, přičemž jak je zmíněno výše, jejich rozdělení je stále uznáváno. Návrhové vzory byly rozděleny tří skupin:

- tvořivé vzory (Creational Patterns),
- strukturální vzory (Structural Patterns),
- behaviorální vzory (Behavioral Patterns).

Následující popis jednotlivých skupin návrhových vzorů vychází přímo z (Gamma, a další, 1994).

Tvořivé návrhové vzory řeší problémy související s vytvářením objektů v systému. Snahou těchto návrhových vzorů je popsat postup výběru třídy nového objektu a zajištění správného počtu těchto objektů. Většinou se jedná o dynamická rozhodnutí učiněná za běhu programu. Mezi tyto návrhové vzory patří:

- Singleton
- Factory
- Factory Method
- Abstract Factory
- Builder
- Prototype

Strukturální návrhové vzory představují skupinu návrhových vzorů zaměřujících se na možnosti uspořádání jednotlivých tříd nebo komponent v systému. Snahou je zpřehlednit systém a využít možností strukturalizace kódu. Mezi tyto návrhové vzory patří:

- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy

Behaviorální návrhové vzory se zabývají chováním systému. Mohou být založeny na třídách nebo objektech. U tříd využívají při návrhu řešení především principu dědičnosti. V druhém přístupu je řešena spolupráce mezi objekty a skupinami objektů, která zajišťuje dosažení požadovaného výsledku. Mezi tento typ vzorů můžeme zařadit:

- Chain Of Responsibility
- Command
- Interpreter

- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template
- Visitor

8.2 Prostředky pro modelování procesů při tvorbě servisně orientovaných systémů

Thomas Erl v (Erl, 2006) uvádí, že jedním z klíčových faktorů při tvorbě servisně orientovaného systému je správně zpracovaný procesní model.

Počátky většího rozšíření procesního modelování spadají do počátku 90. let minulého století, kdy zaznamenal velký vzrůst popularity tzv. Business Process Reengineering (BPR). Tento manažerský přístup je založen na principiálním zhodnocení a radikálním přetvoření stávajících (business) procesů s cílem dosáhnout zlepšení kritických ukazatelů jako jsou náklady, kvalita, rychlost atd.

Z myšlenky BPR vychází Business Process Management (BPM) – řízení business procesů. Jedná se o disciplínu hraničící s managementem a informačními technologiemi, která zahrnuje metody, techniky a nástroje pro návrh, provedení, kontrolu a analýzu operačních procesů, které se týkají lidí, organizací, aplikací, dokumentů a dalších zdrojů informací (van der Aalst, a další, 2003). Termínem *operační proces* rozumíme opakující se procesy, který v rámci organizace probíhají v kontextu její každodenní činnosti. BPM tedy pokrývá aktivity organizace, které mají za cíl řídit a v případě nutnosti zlepšovat její procesy.

8.2.1 Business Process Modeling Notation

Jedním z prvních kroků při využití BPM je fáze mapování (modelování) procesů. Právě v této fázi však dochází k možnému vzniku problému „babylonské věže." Mnohé metodiky totiž využívají vlastních notací pro znázornění procesních diagramů. Tato skutečnost a fakt, že v tomto odvětví chyběl jakýkoliv rozšířený standard, vedla sdružení Business Process Modeling Initiative (BPMI) k zahájení práce na nezávislé notaci pro modelování obchodních procesů – tedy k vytvoření kýženého standardu. Mezi členy BPMI patří i velcí hráči na poli informačních systémů, mj. Adobe Systems, IBM, Intalio, Oracle a SAP. (BPMI, 2008)

Výsledkem práce BPMI je standard Business Process Modeling Notation (BPMN) – notace pro modelování obchodních procesů. Základní vlastností BPMN je naprostá nezávislost na využitém postupu nebo metodice použité při mapování nebo návrhu obchodního procesu. BPMN ve svých produktech podporuje v současné době již více než 40 producentů analytického a podpůrného software pro BPM (BPMI, 2008).

Od roku 2005 jsou další vývojové aktivity okolo BPMN prováděny pod hlavičkou Object Management Group (OMG), protože došlo ke sloučení BPMI a OMG (BPMI, 2008).

Primárním cílem BPMN je poskytnout notaci, která je snadno čitelná pro všechny, kteří s ní přijdou do styku. Jedná se například o obchodní analytiky, kteří vytváří a upravují procesy, vývojáře, kteří jsou odpovědní za jejich implementaci, nebo manažery, kteří kontrolují a řídí jejich průběh. Podobně je BPMN zamýšleno jako společný komunikační prostředek, který zaceluje mezeru, která se mnohdy objevuje mezi oblastmi návrhu obchodních procesů a jejich implementace.

V současnosti existuje mnoho jazyků, nástrojů a metodik pro modelování obchodních procesů. Přijetí BPMN jako standardní notace pomůže sjednotit vyjadřování základních i rozšířených konceptů obchodních procesů. (BPMI, 2008)

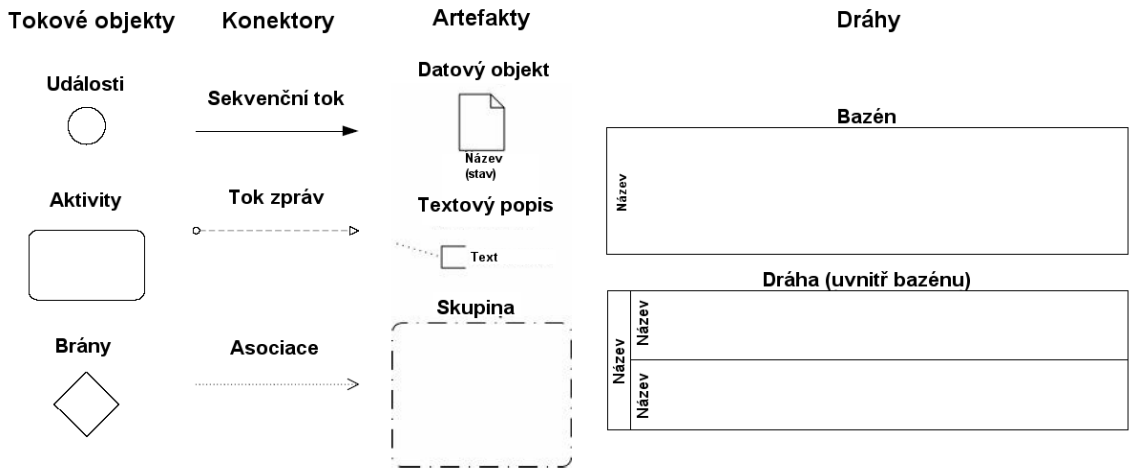
8.2.1.1 Základní prvky BPMN

Diagram obchodního procesu (BPM) se skládá ze základních grafických prvků. Tyto prvky byly vybrány tak, aby byly snadno rozlišitelné a zároveň co nejpodobnější prvkům, které jsou modelujícím běžně známé. Vzhledem k tomu, že BPMN musí vyhovět dvěma takřka protichůdným požadavkům – být jednoduše čitelné a mít schopnost zachytit komplexní informaci – jsou jednotlivé prvky rozděleny do kategorií, v rámci nichž dochází k další diverzifikaci. Při letmém pohledu na diagram lze tedy snadno rozeznat základní typy elementů a tak mu porozumět. (OMG BPMN, 2010)

Díky diverzifikaci v rámci kategorií lze ke každému typu přidat dodatečnou informaci bez toho, aby došlo velké změně diagramu.

Následující popis jednotlivých prvků BPMN čerpá ze specifikace BPMN 2.0 Beta 2 (OMG BPMN, 2010). Základní typy prvků jsou:

- tokové objekty,
- spojovací objekty (konektory),
- dráhy (swimlanes),
- artefakty.



Obr. 8.1. Základní podoby elementů notace BPMN (vlastní zpracování podle (OMG BPMN, 2010)).

8.2.1.2 Tokové objekty

Tokové objekty jsou základní grafické elementy definující chování obchodního procesu. Dělí se na tři druhy – události, aktivity a brány (gateways). Jednotlivé tokové objekty lze spojovat pomocí spojovacích objektů.

Aktivita. Aktivitou rozumíme činnost, která probíhá v rámci obchodního procesu. Aktivita může být atomická nebo složená a podle tohoto hlediska se aktivity dělí na úlohy a podprocesy. Aktivity se značí jako obdélníky se zaoblenými rohy a mohou se provádět jen jednou nebo opakovaně v cyklu.

- *Úloha* je atomická aktivita uvnitř procesu. Úloha se používá tam, kde dále nechceme nebo nemůžeme na aktivitu pohlížet na větší úrovni detailů.
- *Podproces* je složená aktivita uvnitř procesu. Může být podrobněji rozebrána pomocí podaktivit. Podproces se může v diagramu nacházet buď ve sbalené, nebo rozbalené formě. Sbalenou formu poznáme podle symbolu „plus“ v entitě podprocesu, která nám říká, že se jedná o podproces na nižší úrovni detailů.

Událost. Událostí rozumíme něco, co se „stane“ v průběhu procesu. Tyto události ovlivňují tok procesu a obvykle mají příčinu (trigger, spoušť) nebo důsledek (výsledek). Události zakreslujeme jako kruhy s prázdným prostředkem, což umožňuje pomocí dalších vnitřních značek rozlišit rozdílné spouštěče a výsledky. Podle účinku na tok rozlišujeme tři druhy událostí – start, průběžnou událost a konec.

- *Událost Start* indikuje počátek procesu. Existuje několik spouštěčů, které odlišují specifické okolnosti počátku procesu. Prázdný element Start se používá k určení počátku podprocesu nebo v místě, kde není počátek jasně definován.
- *Průběžné události* nastávají mezi započetím a ukončením procesu. Mohou být použity jako běžný element v rámci toku, ale i připojeny k hranici aktivity. Události umístěné v běžném toku reprezentují to, co

se udává během normálního průběhu procesu. Mohou představovat reakci na událost nebo i vznik události. Události připojené k aktivitě dávají na vědomí, že v případě výskytu události dojde k přerušení aktivity. Lze je připojit jak k úloze, tak k podprocesu. Obvykle se používají pro ošetření chyb, výjimek a kompenzace.

- *Událost Konec* ukazuje, kde je ukončen běh procesu. Existuje několik různých „výsledků“, které ukazují specifické okolnosti, které proces ukončují.

Brána. Brána je element, který je používán pro kontrolu sekvenčního toku (jeho rozdělování a sbíhání). Všechny typy bran se značí jako kosočtverce. Rozdílné vnitřní značky určují odlišnosti jejich chování. Všechny brány tok jak rozdělují, tak spojují. Brány přímo určují místa, kde je potřebná kontrola toku procesu.

- *Vylučovací brány* (rozhodnutí) jsou místa v toku obchodního procesu, kde se lze vydat jedním z několika směrů. Jedná se o klasickou „křižovatku“ uvnitř procesu. Z této brány lze pokračovat jen jednou cestou. V závislosti na způsobu rozhodnutí se rozlišují vylučovací brány založené na datech a na událostech.

  Vylučovací brána založená na datech je nejčastěji používaným druhem brány. Může být označena pomocí vnitřní značky „X“, ale obvykle se používá bez ní. Je založena na větvení toku pomocí splnění dané podmínky.

  Vylučovací brána založená na událostech představuje rozdělovací bod v procesu, kde jsou alternativy určeny pomocí událostí, ke kterým dojde v tomto bodě. K identifikaci tohoto druhu brány se jako vnitřní značky využívá značky elementu „násobná průběžná událost.“ Událost uvedená v toku za branou je rozhodovacím faktorem. První aktivovaná událost vyhrává a tok směřuje jejím směrem.

- *Paralelní brány* jsou místa v procesu, kde jsou definovány několikeré paralelní cesty. Tyto brány jsou identifikovatelné pomocí vnitřní

značky „+“ uvnitř symbolu brány. Používají se jak pro rozdělení tak spojení cesty.

8.2.1.3 Spojovací objekty

Spojovací objekty, nebo také „konektory“, jsou elementy, které slouží k propojování tokových objektů. Podobně jako tokové objekty se i spojovací objekty dělí na tři druhy – sekvenční tok, tok zpráv a asociace.

Sekvenční tok. Ukazuje pořadí, v jakém budou v rámci procesu vykonávány jednotlivé aktivity. Sekvenční tok nemůže překročit hranice podprocesu nebo bazénu. Může mít na výstupu z aktivity definovanou podmínku. Taková aktivita pak musí mít přinejmenším dva výstupní sekvenční toky. Při splnění podmínky probíhá tok danou cestou dále. Podmínkový sekvenční tok se označuje kosočtvercem na straně výstupu ze zdrojové aktivity.

Tok zpráv. Využívá se k zobrazení toku zpráv mezi dvěma participanty procesu. V BPMN se pro zobrazení participantů využívají oddělené bazény. Tok zpráv se může dotýkat buď hranice bazénu nebo entity v tomto bazénu. Tímto tokem nelze spojit dvě entity v jednom bazénu.

Asociace. Využívá se k asociaci dat, informací a artefaktů s tokovými objekty. Ukazuje, jak data vstupují nebo vystupují z aktivity. K asociaci lze připojit textový popis.

8.2.1.4 Dráhy

BPMN využívá pro snazší členění a organizaci aktivit tzv. konceptu „plaveckých drah.“ Tato skupina obsahuje dva objekty – bazén a dráhu.

Bazén. Představuje participanty interaktivního B2B procesního diagramu. Participantem může být buď obchodní role (prodejce, kupující) nebo obchodní entita (konkrétní jedinec, společnost). Bazén může být pro okolí „černou skříňkou“ nebo může obsahovat viditelné procesy. Interakce mezi jednotlivými bazény je zajištěna pomocí toků zpráv. Sekvenční toky

nesmí opustit hranice jednotlivých bazénů. Procesy jsou tedy bazény plně ohraničeny.

Dráha. Dráhy představují části v rámci jednoho bazénu. Často představují role v rámci organizace, ale dají se využít na prezentaci jakýchkoliv charakteristik procesu. Sekvenční tok může procházet skrz hranice drah.

8.2.1.5 Artefakty

Poslední skupinou jsou artefakty, které jsou využívány k podání dodatečné informace o procesu. Existují tři standardizované artefakty, ale návrháři a tvůrci modelovacích nástrojů smí volně doplňovat množinu artefaktů podle svých potřeb. Musí samozřejmě dodržet podmínku, že vytvořený artefakt nelze zaměnit s již existujícím elementem v rámci BPMN.

8.2.2 Business Object Relation Modeling

Metoda BORM byla vyvíjena postupně od roku 1993. Původně pouze jako nástroj pro tvorbu objektově orientovaných softwarových systémů. BORM je navržen jako metoda, která pokrývá všechny fáze vývoje softwaru. Velká pozornost je věnována počátečním projektu a postupům, jak najít objekty v zadaném problému a zkontrolovat jejich správnost. Modelování obchodních procesů je jedním z klíčových prvků této metody.

Metodě BORM jsou věnovány následující publikace, které budou v této kapitole využívány jako hlavní zdroje - (Knott, a další, 2003), (Knott, a další, 2006). Využití metody BORM v oblasti procesního modelování organizace rozebírá (Merunka, a další, 2010).

Projekt zpracovávaný pomocí metody BORM se dá rozdělit do dvou základních fází – etapa expanze a etapa konsolidace.

Fáze expanze začíná analýzou modelu obchodních (nebo také byznys) objektů. Dochází zde k hromadění informací potřebných pro vytvoření aplikace. Stádium expanze končí s dokončením analytického konceptuálního

modelu, který na logické úrovni reprezentuje požadované zadání a v abstraktní úrovni popis jeho řešení.

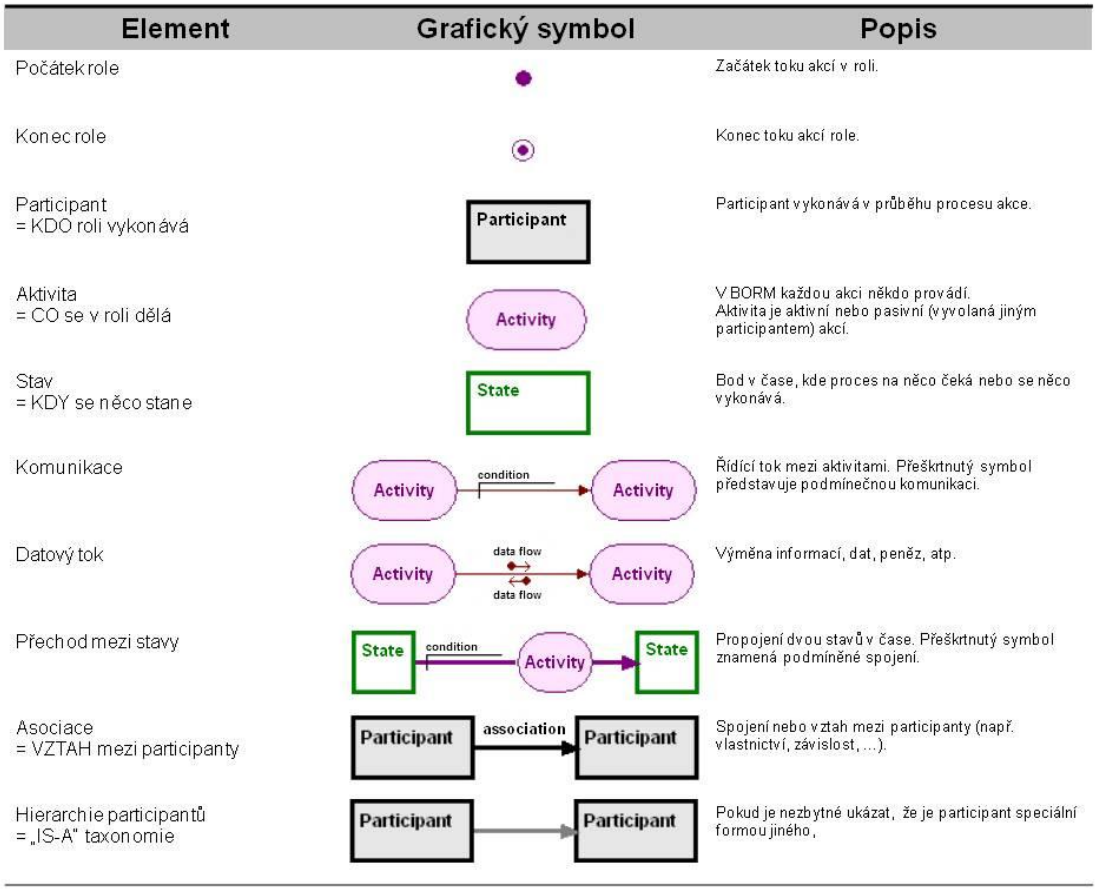
Zbývající fáze od konceptuálního modelu až k finálnímu systému složenému ze softwarových objektů se označují jako *stádium konsolidace*. Je tomu tak proto, že v těchto etapách se model, který je produktem předchozí expanze, postupně stává fungujícím programem. To znamená, že na nějakou myšlenkovou „expanzi" zadání zde již není prostor ani čas. V tomto stadiu se také počítá s tím, že od některých idejí z expanzního stádia bude třeba upustit vzhledem k časovým, kapacitním, implementačním nebo i intelektuálním schopnostem – odtud tedy název tohoto stadia. Takto odstraněná informace však může být v budoucnu základem analýzy nové verze systému. (Merunka, 2008) (Knott, a další, 2003)

8.2.2.1 Procesní diagram metody BORM

Během fáze expanze je zpracováván model obchodních procesů problémové domény. Ke znázornění tohoto modelu slouží procesní diagram metody BORM. Na rozdíl od procesních diagramů s notací BPMN, které nejsou součástí žádné konkrétní metody, je procesní diagram BORM přímou součástí této metody a má formální základy. (Knott, a další, 2003)

Zatímco procesní diagram zakreslený pomocí BPMN je de facto flow chart, procesní diagram metody BORM je založen na principu konečného automatu. Tento fakt sám o sobě otevírá možnosti sofistikovaného ověření navrženého procesního modelu například pomocí simulace. Tvorba procesních diagramů metody BORM je možná s využitím patřičné knihovny například v modelovacím nástroji MetaEdit (v tomto nástroji je metoda demonstrována v (Knott, a další, 2000)). Přímo pro metodu BORM pak vznikl modelovací nástroj Craft.CASE (Craft.CASE ltd.), který zahrnuje všechny etapy tvorby projektu a umožňuje verifikaci a simulaci procesních diagramů.

Notace procesního diagramu metody BORM je v některých rysech podobná notaci BPMN, ale obsahuje i klíčové rozdíly. Jednotlivé prvky notace procesních diagramů BORM jsou vyobrazeny na obrázku 8.2.



Obr. 8.2. Základní prvky notace procesních diagramů metody BORM (do češtiny přeloženo z (Merunka, a další, 2010))

8.2.2.2 BORM z pohledu MDA

MDA (Model-Driven Approach) je jednou ze současných metodik pro vývoj software. Poskytuje vodítka pro strukturování specifikací, které jsou vyjadřovány pomocí postupné transformace modelů. Byla představena OMG v roce 2001 a je jednou z nejrozšířenějších metod postavených na využití modelovacího jazyka UML.

Podle (Merunka, a další, 2010) lze BORM považovat za speciální podobu MDA (OMG MDA, 2010). V MDA terminologii můžeme BORM popsat takto:

- Model CIM (Computer-Independent Model) je podle metody BORM vizualizací prostředí, ve kterém je projekt vykonáván. Primárně se zaobírá modely obchodních procesů. Jeho cílem je porozumět problému, popsat ho a najít jeho řešení. Korektně zpracovaný CIM model umožňuje správně popsat všechny předpoklady a požadavky na tvorbu informačního systému. Tato část metody BORM v podobě diagramu obchodních procesů bude názorně předvedena v následující kapitole.
- Model PIM (Platform-Independent Model) v rámci metody BORM představuje vizualizaci požadovaného informačního systému z pohledu softwarového inženýrství. V tomto modelu hraje zásadní roli modelovací jazyk UML. Pro transformaci do podoby konceptuálního modelu v UML byla stanovena pravidla (Knott, a další, 2000) a je možné využít i automatických mechanismů nástroje Craft.CASE (Merunka, a další, 2008).
- Model PSM (Platform-Specific) je revidovaná podoba modelu PIM, která je ovšem zaměřena na konkrétní implementační prostředky, může popisovat specifické vlastnosti cílového prostředí, využití stávajících artefaktů IT infrastruktury, atd. Taktéž v tomto případě existují pravidla a nástroje pro převod PIM modelu na tento model (Merunka, a další, 2008).

8.2.2.3 Ukázkový příklad použití metody BORM pro modelování procesů

Na tomto ukázkovém příkladu budou demonstrovány základní vlastnosti metody BORM ve fázi expanze. V průběhu popisu analýzy budou zmiňovány další vlastnosti metody BORM, hlavním zdrojem je publikace (Merunka, 2008).

Definice problémové domény

Modelovou organizací je letecká společnost FunFly, která chce zvýšit svojí konkurenceschopnost zavedením elektronického systému pro rezervace pilotů, kteří využívají jejích služeb. Firma také výhledově rozšiřuje svoje

služby o možnost leteckého výcviku a požaduje tedy, aby nový systém bral i ohled na patřičné změny v procesech, protože stávající podoba vyžadovala nutnost rezervace instruktora pro výcvikový let samostatně.

V organizaci v současné (AS-IS) podobě není žádný podobný systém provozován a veškeré rezervace jsou evidovány ručně a je nutné je provádět telefonicky nebo osobně.

Zjištění požadavků na funkčnost – metoda OBA

Metoda Object Behavior Analysis je technika sloužící pro k získávání strukturovaných podkladů ze zadání pro potřeby konstrukce prvotního objektového modelu. Metoda OBA vznikla počátkem 90. let minulého století a jejími autory jsou Kenneth S. Rubin a Adele Goldberg. (Rubin, a další, 1992) (Knott, a další, 2000)

BORM využívá metodu OBA jako jeden ze svých nástrojů. Jednotlivé kroky OBA v BORM jsou následující:

- **rozpoznání procesů**. V tomto kroku se na základě provedeného interview sestaví seznam požadovaných funkcí systému a klíčové objekty v systému. Jedná se vesměs o textové popisy. Cílem tohoto kroku je nejen zahájit stavbu modelu, ale vymezit zadání v rámci možného širšího kontextu řešeného problému.
- **rozpoznání plánování scénářů jako detailního popisu již rozpoznaných funkcí a popisy vlastností objektů**. V tomto kroku se u každého scénáře rozlišuje původ procesu, vlastní popis procesu, participující objekty a popis výsledku procesu.
- **definování vztahů mezi objekty navzájem a mezi objekty a procesy pomocí modelových karet**. V tomto kroku se pro každý rozpoznaný objekt z předchozího kroku vytvoří jeho modelová karta, která obsahuje jméno objektu, seznam aktivit objektu a s ním související seznam s modelovaným objektem spolupracujících objektů.

- **modelování procesů**. V tomto kroku se pro každý rozpoznaný objekt s pomocí informací v tabulce scénářů a modelových kartách sestaví životní cyklus objektu jako sled jeho stavů a přechodů mezi těmito stavy v podobě procesního diagramu.
- **verifikace a validace**. Zde se kontroluje shoda mezi diagramy, tabulkami a skutečnými požadavky na systém. K tomu slouží dva nástroje. Jedním z nich je datový model obsahující skutečná data pomocí nichž lze prověřit správnost návrhu. Druhým nástrojem je simulátor procesů, který dovoluje „sehrát a vyzkoušet" proces znázorněný diagramy. (Merunka, 2008)

Rozpoznání procesů

Pomocí interview byly identifikovány následující funkce. Procesy označené jako AS-IS jsou stávající, procesy TO-BE jsou procesy očekávané od nového systému.

Název	Popis	Stav
2	Piloti žádají o letadla	AS-IS
3	VLP přiděluje letadla	AS-IS
4	Mechanici se starají o letadla	AS-IS
5	Piloti používají letadla na výcvik	AS-IS
6	Žák žádá o instruktora	AS-IS
7	VLP přiděluje instruktora	AS-IS
8	VLP plánuje provoz	AS-IS
9	Žák žádá o letadlo	AS-IS
14	Pilot ruší rezervaci	AS-IS
New 1	Piloti se přihlašují na provoz	TO-BE

New 10	Piloti/žáci žádají o letadla	TO-BE
New 11	VLP přiděluje letadla pilotům	TO-BE
New 12	VLP přiděluje letadla s instruktorem žákům	TO-BE
New 13	Mechanici předávají a přebírají letadla	TO-BE
New 14	Administrace pilotů	TO-BE

Tabulka 8.1. Seznam funkcí existujících a požadovaných

Participanti

Na základě interview byli identifikováni následující participanti procesů.

Participant	Popis
Instruktor	Vyučuje žáky.
Pilot	Využívá letadlo, nepotřebuje instruktora.
Žák	Využívá letadlo k výcviku, potřebuje instruktora.
VLP	Organizuje provoz.
Mechanik	Stará se o letadla.
Letadlo	Letadlo používané k vykonání letu.

Tabulka 8.2. Seznam participantů v projektu.

Definice scénářů

Jednotlivým nalezeným funkcím odpovídají patřičné scénáře. Zatímco v první fázi definice funkcí byly tyto jen slovně popsány, v případě scénářů je jejich popis díle rozvinut, včetně vytvoření seznamu jejich účastníků a popisu jejich podílu na daném scénáři.

2	Derived from: 2, 3, 4	as is
initiation: Rozhodnuti VLP		*roles:* **Letadlo** **Mechanik** cooperates **Pilot** is responsible **VLP** approves
action: Prideleni letadla pilotovi na let		
result: Pilot ma prideleno letadlo na let		
attached diagram:		

Obr. 8.3. Scénář procesu 2 - "Přidělení letadla pilotovi na let", podoba AS-IS

Jak je vidět z obrázku 8.3, který znázorňuje výstup ze systému Craft.CASE, má proces tři fáze – iniciace, akce a výsledek. Proces je iniciován rozhodnutím VLP o povolení letu. Akcí VLP je přidělení letadla pilotovi. Výsledkem procesu je přidělené letadlo pilotovi zaznamenané v evidenci. Procesu se účastní čtyři participanti v různých rolích. Letadlo je objektem procesu, Mechanik se na procesu podílí, Pilot je za něj odpovědný (má přiděleno letadlo a má za něj tudíž zodpovědnost) a VLP přidělení schvaluje.

Diagram tohoto procesu vychází z nalezených funkcí 2, 3, 4, které lze nalézt v tabulce 8.1.

new 6	Derived from: new 10, new 11, new 12, new 13, 5	to be
initiation: Rozhodnuti VLP		*roles:* **Instruktor** cooperates **Letadlo** **Mechanik** is responsible **Pilot** is responsible **VLP** approves **Zak** is responsible
action: Prideleni letadla pilotovi/zakovi na let		
result: Pilotu je prideleno letadlo na provedeni letu. Pokud se jedna o zaka, je mu pridelen i instruktor.		
attached diagram:		

Obr. 8.4. Diagram téhož procesu v podobě TO-BE, scénář New 6 - "Přidělení letadla pilotovi/žákovi"

Jak je patrno z obrázku 8.4, dochází v TO-BE podobě ke sloučení dvou původních scénářů do jednoho. V seznamu rolí pak přibývá nově Instruktor a Žák.

Modelové karty

Modelové karty určují, v jakém vztahu k ostatním participantům je daný participant v rámci všech scénářů, jichž se účastní.

Collaborators:	Instruktor	Letadlo	Mechanik	Pilot	Zak
new 6 (approves)	cooperates	X	is responsible	is responsible	is responsible
new 7 (is responsible)				X	X
new 8 (performs)					
1 (is informed)	X	X	is informed	X	is responsible
2 (approves)		X	cooperates	is responsible	
3 (cooperates)				performs	
4 (approves)	cooperates	X	is informed		is responsible

Obr. 8.5. Modelová karta participanta Vedoucí letového provozu (VLP).

Jak je vidět z modelové karty participanta VLP na obrázku 8.5, účastní se celkem 7 scénářů a spolupracuje na nich se všemi ostatními participanty. X v modelové kartě značí, že se participant účastní daného scénáře v implicitní (default) roli a není mu tedy nastavena některá z rolí specifických.

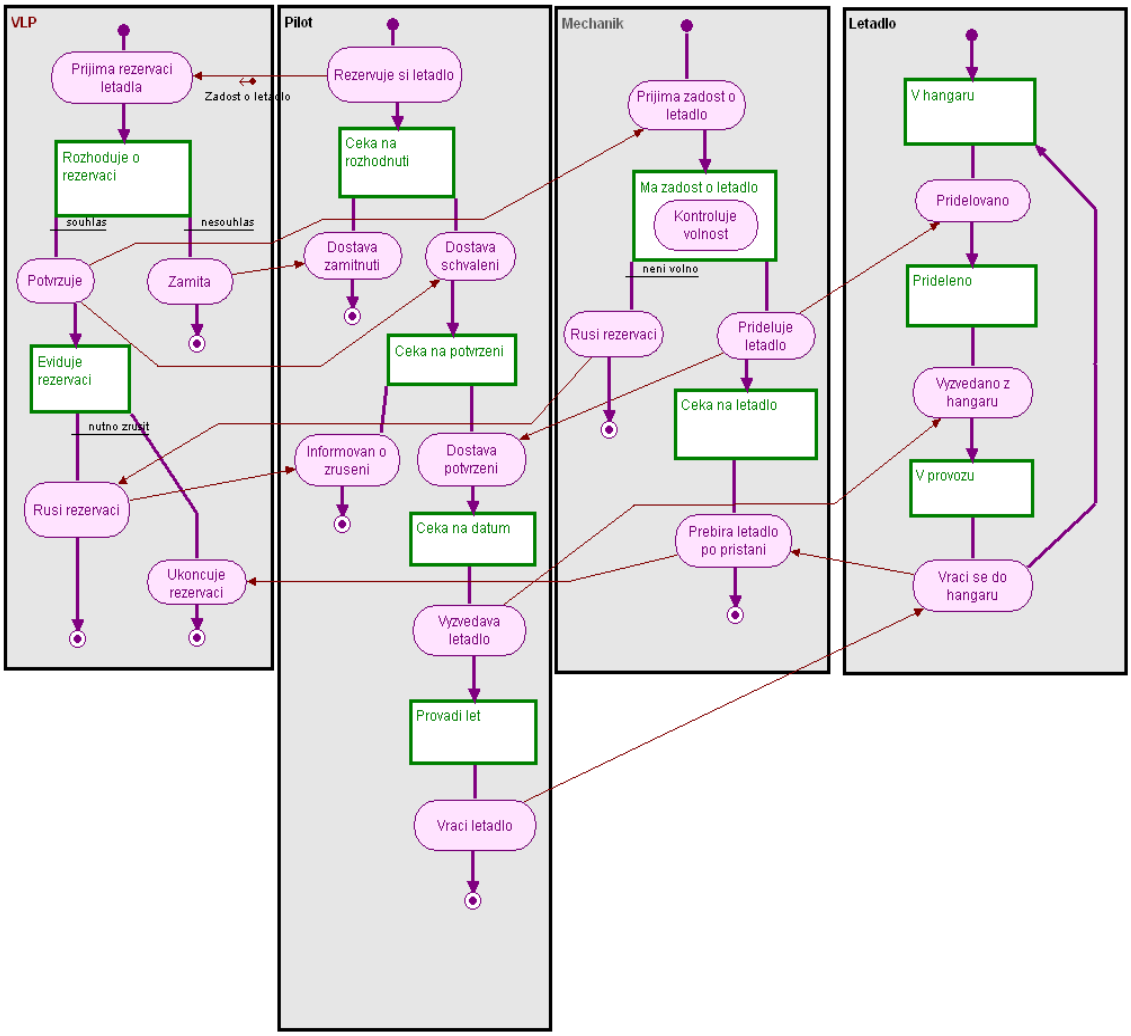
Modelování procesů

V této fázi dochází na základě definic scénářů a informací z modelových karet k vytvoření tzv. procesních diagramů, které vyjadřují životní cyklus všech objektů.

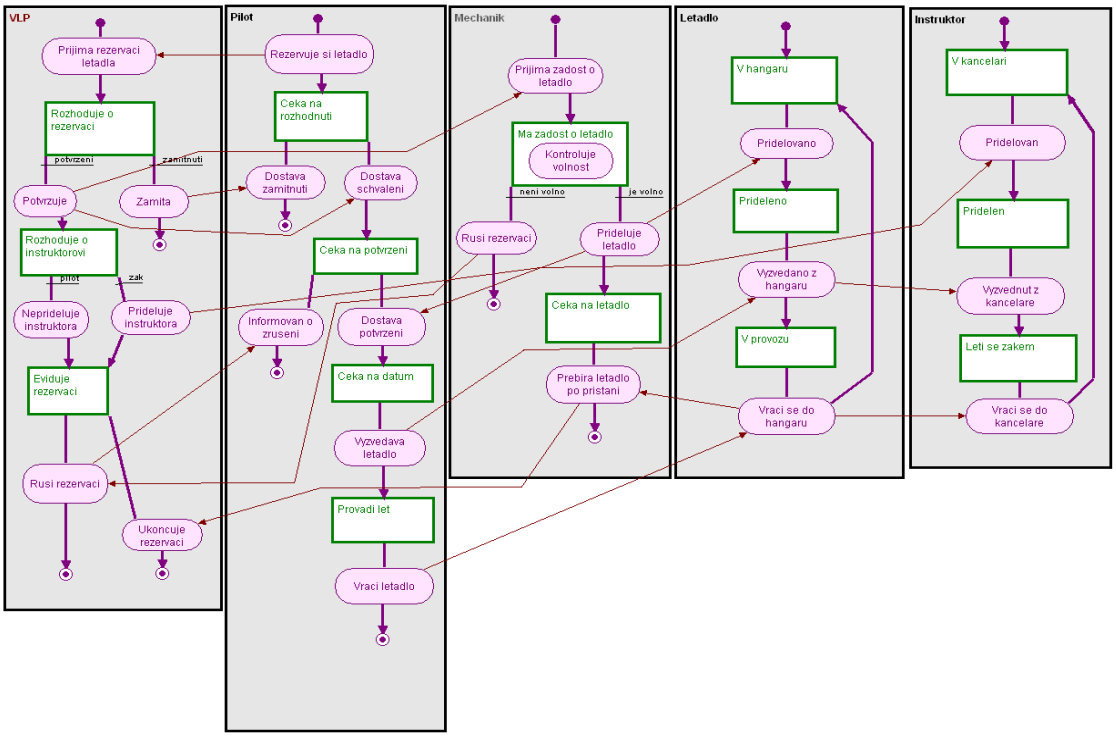
Vyplněná kolečka značí počátek činnosti pro daného participanta, kružnice s kolečkem pak konec jeho činnosti. Obdélník označuje stav, obdélník s kulatými rohy pak představuje aktivitu (více viz obrázek 8.2).

Ověření správnosti

Ověření správnosti procesního diagramu je možné pomocí simulačního nástroje v aplikaci Craft.CASE. Oba dále uvedené diagramy prošly simulací úspěšně a jsou tedy validním znázorněním daných procesů.



Obr. 8.6. Procesní diagram rezervace, přidělení letadla a provedení letu v podobě AS-IS pro pilota



Obr. 8.7. Procesní diagram pro TO-BE podobu procesu rezervace, přidělení letadla a vykonání letu pro pilota i žáka.

8.3 Service Oriented Modeling Framework

Service-Oriented Modeling Framework (SOMF) je univerzální prostředek vytvořený Michaelem Bellem a dále udržovaný a rozvíjený pod hlavičkou Methodologies Corporation, jehož cílem je poskytnout modelovací jazyk využitelný k návrhu jakékoliv aplikace či aplikačního prostředí a to jak lokálního, tak distribuovaného (Bell, 2008). SOMF lze využít i k modelování rozsáhlého SOA prostředí, včetně řešení v podobě cloudů (Bell, 2010). Aktuálně se SOMF nachází ve verzi 2.1. (Sparx, 2011)

SOMF a jeho notace byla vybrána firmou Sparx a je nyní součástí jejího modelovacího nástroje Enterprise Architect (Sparx, 2011).

8.3.1 Modelové generace

Podobně jako metody pro mapování procesů umožňuje SOMF členit modely do časových období. Přidává však o jednu časovou úroveň navíc – pohled do minulosti. Model zpracovaný pomocí SOMF se tak může nacházet

v následujících časových obdobích, nazývaných také modelové generace (Bell, 2008):

- **Used-to-Be**. Schéma softwarových komponent a jim příslušného prostředí v podobě, ve které byly nasazeny, konfigurovány a využívány v minulosti.
- **As-Is**. Softwarové komponenty a jim příslušné prostředí v podobě, v jaké jsou aktuálně používány.
- **To-Be**. Návrh softwarových komponent a jim příslušného prostředí, které budou nasazeny, konfigurovány a používány v budoucnu.

8.3.2 Transformační modely

SOMF nabízí osm implementačních modelů, taktéž známých jako Bellovy transformační modely (Bell, 2008). Každý z modelů popisuje určitou část vývoje projektu. Těchto osm modelů bývá spojováno s devátým, kontrolním modelem, který by měl být využíván pro řízení zbylých osmi modelů.

Implementační modely SOMF podle (Bell, 2008) jsou:

- objevný model (Discovery Model),
- analytický model (Analysis Model),
- návrhový model (Design Model),
- model technické architektury (Technical Architecture Model),
- konstrukční model (Construction Model),
- model zajištění kvality (Quality Assurance Model),
- operační model (Operations Model),
- model business architektury (Business Architecture Model),
- kontrolní model (Governance Model) – model zaštiťující ostatní modely.

8.3.3 Aktiva SOMF

V kontextu SOMF bývají jednotlivé softwarové komponenty označovány jako „aktiva“. Softwarové aktivum může představovat např. jakýkoliv objekt, softwarový modul, knihovna, aplikace, proces, databáze či v ní uložená procedura nebo trigger, ESB, již existující („legacy“) implementace, webová služba a další. Všechny tyto softwarové entity se mohou v kontextu SOMF nazývat „služba“ (Bell, 2008).

V souvislosti s aktivy SOMF, potažmo službami, se hovoří o jejich jemnosti, případně hrubosti. Čím jemnější je služba, tím je specializovanější a její funkcionalita je omezena jen na určitou činnost (Bell, 2008).

Služby se podle svých atributů mohou dělit do těchto skupin:

- Atomické služby – dále nedělitelné softwarové komponenty. Atomická služba se obvykle dále nedekomponuje a její funkcionalita je natolik omezená, že nevyžaduje rozčlenění na jemnější prvky.
- Složené služby – agregují menší a jemněji členěné služby. Tato hierarchická podoba služby je charakteristická hrubší strukturou. Složená služba může agregovat jak atomické, tak další složené služby.
- Klastry služeb – jedná se o kolekce služeb, které jsou seskupeny kvůli svým společným vlastnostem (ať po stránce technické nebo podobné funkcionality). Klastr služby jak sdružuje, tak jich využívá k řešení dalších problémů. Může obsahovat jak atomické, tak složené služby.
- Cloud služeb – seskupení služeb, které je realizováno pomocí prostředků cloud computingu.

8.3.4 Modely a notace SOMF

SOMF nabízí 360° pohled na životní cyklus vývoje software. Počínaje konceptuální fází, přes návrh a nasazením v produkčním prostředí konče. Pro tyto účely bylo vytvořeno šest modelů s patřičnými notacemi (Methodologies Corporation, 2011a). Jedná se o:

1. Service-Oriented Conceptualization Model (SO konceptuální model)
2. Service-Oriented Discovery and Analysis Model (SO analytický model)
3. Service-Oriented Business Integration Model (SO integrační model)
4. Service-Oriented Logical Design Model (SO logický model)
5. Service-Oriented Software Architecture Model (SO model softwarové architektury)
6. Cloud Computing Toolbox Model (model pro cloud computing)

V následující části budou představeny notace analytického a logického modelu.

8.3.5 Notace analytického modelu SOMF

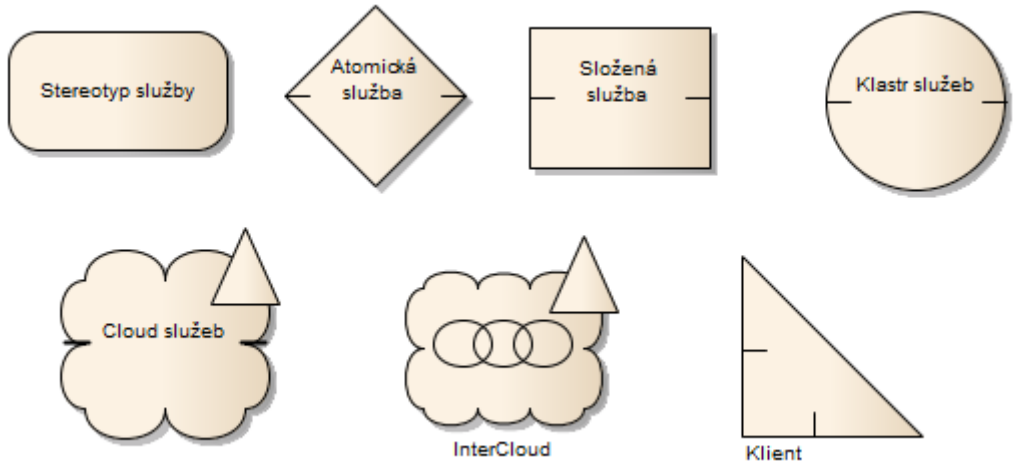
Hlavním zdrojem této kapitoly je oficiální dokumentace analytického modelu SOMF verze 2.1, publikovaná v (Methodologies Corporation, 2011a).

Notace analytického modelu SOMF sestává ze základních prvků modelu v podobě aktiv SOMF a množiny konektorů pro vyjádření vazeb mezi nimi.

Na obrázku 8.8 jsou zobrazena základní aktiva SOMF v notaci analytického modelu. Atomická služba, složená služba, klastr služeb a cloud služeb byly popsány v kapitole 8.3.3. Popis aktiv, která nebyla součástí této kapitoly je uveden níže:

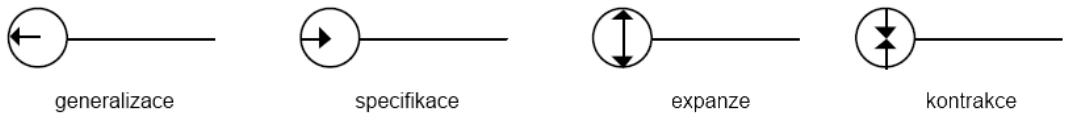
- Stereotyp služby – obecná služba, u které není identifikována struktura.

- InterCloud – představuje pojem „cloud cloudů“, tj. cloud nadřazený skupině dalších cloudů, které spolupracují a tvoří komplexnější řešení.
- Klient – entita představující zákazníka. Může představovat jak uživatele jednotlivých služeb, tak celých aplikací.



Obr. 8.8. Aktiva SOMF v notaci analytického modelu
(podle (Methodologies Corporation, 2011a))

Pro vyjádření vztahu mezi aktivy jsou v notaci definovány konektory. Ty se dělí do dvou skupin – na kontextové a strukturální.

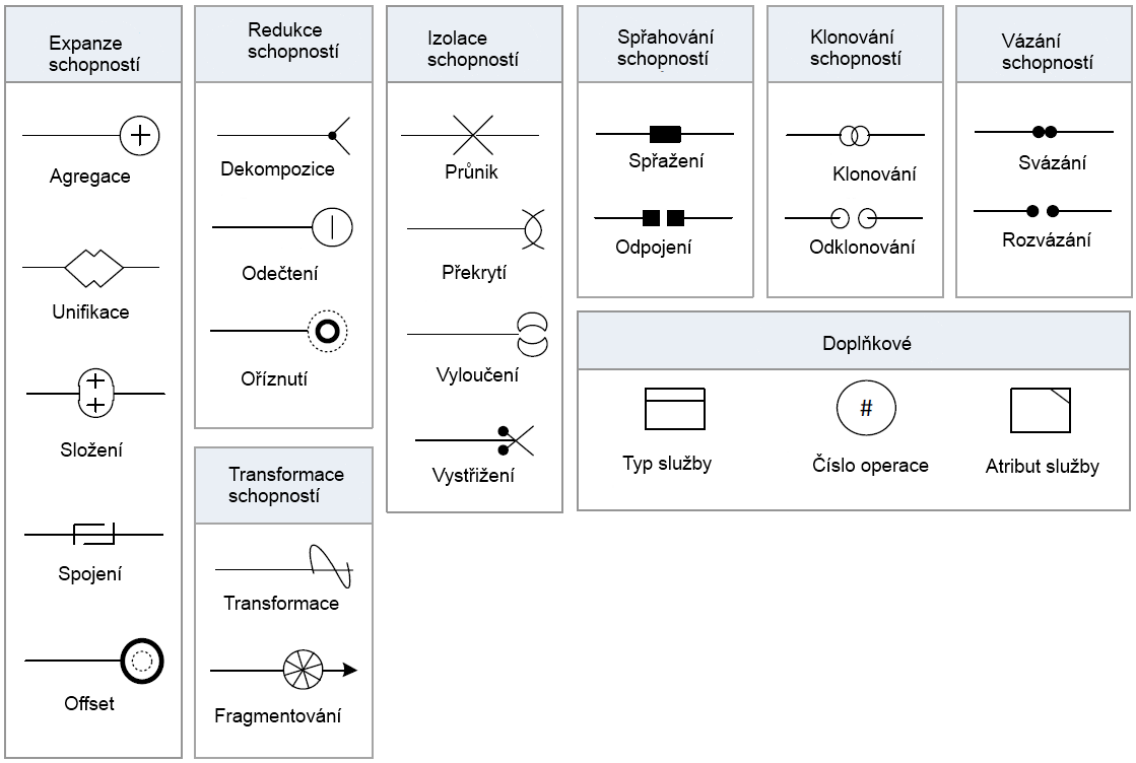


Obr. 8.9. Kontextové konektory
(převzato a přeloženo z (Methodologies Corporation, 2011a))

Do skupiny kontextových konektorů vyobrazených na obr. 8.9 patří:

- generalizace – zvětšuje úroveň abstrakce služby, záběr jejích operací a schopností;
- specifikace – opak generalizace, omezuje úroveň abstrakce služby, záběr jejích operací a schopností;
- expanze – rozšíření architekturálních a technologických schopností;

- kontrakce – opak expanze, omezení architekturálních a technologických schopností.



Obr. 8.10. Strukturální konektory
(převzato a přeloženo z (Methodologies Corporation, 2011a))

Význam vybraných strukturální konektorů z obr. 8.10 je pak následující:

- agregace – vloží jemnější aktivum do hrubšího aktiva,
- unifikace – spojí dvě nebo více aktiv do jednoho,
- složení – seskupí dvě nebo více služeb do skupiny za účelem spolupráce,
- spojení – seskupí dvě nebo více služeb za účelem poskytnutí stálého nebo pevného řešení,
- dekompozice – vyčlení aktivum z jiného aktiva,
- odečtení – ukončení činnosti aktiva (deaktivace),
- transformace – změní strukturu aktiva na jinou,
- fragmentace – rozdělí aktivum na skupinu jemných aktiv. Původní aktivum je potom deaktivováno,

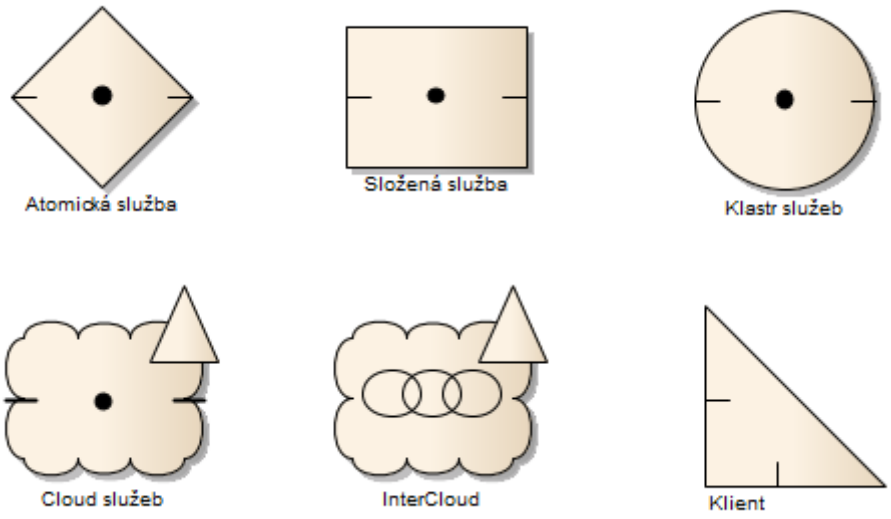
- průnik – průnik dvou aktiv,
- spřažení – spojení dvou aktiv,
- odpojení – oddělení dvou aktiv,
- klonování – vytvoření duplikátu aktiva,
- odklonování – zruší vztah klonování mezi dvěma aktivy,
- svázání – vytvoření formálního vztahu mezi dvěma aktivy,
- rozvázání – zrušení vztahu mezi dvěma nebo více aktivy.

8.3.6 Notace logického modelu SOMF

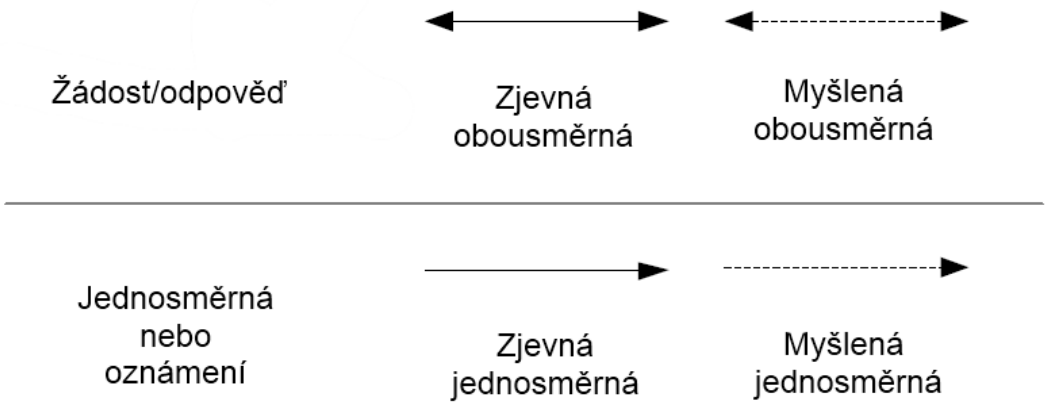
Hlavním zdrojem této kapitoly je oficiální dokumentace logického modelu SOMF verze 2.1, publikovaná v (Methodologies Corporation, 2011b).

Obdobně jako notace analytického modelu, se i notace logického modelu SOMF sestává ze základních prvků modelu v podobě aktiv SOMF a množiny konektorů pro vyjádření vazeb mezi nimi.

Oproti analytickému modelu však v množině aktiv chybí stereotyp služby. Grafická podoba jednotlivých aktiv je znázorněna na obr. 8.11. Popisy jednotlivých aktiv jsou totožné s popisy uvedenými v kapitolách 8.3.3 a 8.3.5.



Obr. 8.11. Aktiva v notaci logického modelu SOMF
(podle (Methodologies Corporation, 2011b))



Obr. 8.12. Konektory logického modelu SOMF – komunikace (podle (Methodologies Corporation, 2011b))

Na obr. 8.12 jsou vyobrazeny konektory logického modelu SOMF, které představují různé druhy komunikačních toků mezi službami. Jednotlivé komunikační toky jsou popsány následovně:

- zřejmá obousměrná komunikace – znázorňuje dvousměrnou komunikaci podle vzoru žádost/odpověď. Typická situace je žádost klienta a odpověď služby. Slovo "zřejmý" v názvu značí přímé spojení mezi službou a jejím konzumentem, které není narušeno žádnou další softwarovou entitou;
- zřejmá jednosměrná komunikace – jednosměrné předání zprávy, které může mít původ jak na straně konzumenta, tak na straně služby. Od příjemce není vyžadována odpověď. Slovo "zřejmý" v názvu značí přímé spojení mezi službou a jejím konzumentem, které není narušeno žádnou další softwarovou entitou;
- myšlená obousměrná komunikace – dvousměrná komunikace mezi klientem a službou. Slovo "myšlená" označuje komunikační kanál, který je zachytáván prostředníkem, který se stará o doručení zprávy na cílové místo;
- myšlená jednosměrná komunikace – jednosměrný komunikační kanál. Slovo "myšlená" naznačuje, že je komunikace zachytávána prostředníkem, který se stará o její doručení na cílové místo.

9 Kritika výchozí stavu

Bouřlivý vývoj technologií souvisejících se servisně orientovanou architekturou vedl ke vzniku různých technických řešení a metod, které jsou šité na míru jednotlivým platformám a řešení od různých výrobců (Štumpf, 2006). Mnozí autoři zpracovávající témata související s oblastí servisně orientovaného paradigmatu vyjádřili potřebu tuto oblast formálně sjednotit, např. Lhotka v (Lhotka, 2009) či van der Aalst, který tuto problematiku prezentoval i ve své keynote na konferenci EOMAS 2010. Část keynote je publikována v (van der Aalst, 2010).

Snahy o formalizaci SOA již několik let probíhají, většinou v rámci specifických problémových domén. Výsledky těchto snažení byly publikovány např. v (van der Aalst, a další, 2009), (Margaria, 2011), (Li, a další, 2009).

Autor této práce navrhuje pro formální popis prvků servisně orientovaného paradigmatu analogických k prvkům objektového modelu využít existujících prostředků pro popis objektově orientovaných systémů, např. v podobě objektového kalkulu podle Merunky (Merunka, a další, 2007).

10 Analogie vybraných vlastností objektově a servisně orientovaného paradigmatu

10.1 Formální aparát

Formální aparát, který je využíván v této práci, je postaven na aplikaci λ-kalkulu na objektové modelování. Základy tohoto objektového kalkulu jsou definovány v publikaci (Merunka, a další, 2007), na níž se podílel i autor této disertační práce, a dále v (Merunka, 2008). Navrhovaná rozšíření tohoto aparátu pro další oblasti objektového modelování byla autorem této disertační práce publikována v (Brožek, 2008). Aplikace na oblast servisně orientovaného paradigmatu je jedním z přínosů této disertační práce.

10.1.1 Operace

V tabulce 10.1 jsou uvedeny základní operace využívaného objektového kalkulu a jim náležející symboly.

Symbol	Operace
◅	zaslání zprávy
$\overset{\text{substituce}}{\Longrightarrow}$	záměna výrazu za jeho jinou podobu (rozepsání)
$\overset{\text{konverze}}{\Longrightarrow}$	náhrada proměnné jinou proměnnou (α-konverze)
$\overset{\text{redukce}}{\Longrightarrow}$	dosazení do výrazu (β-redukce)
$\Longleftarrow$	návrat výsledku operace do proměnné

Tabulka 10.1. Základní operace využívaného kalkulu

10.1.2 Specializované konstrukce

Pomocí využívaného aparátu lze kromě základních operací zaznamenat i komplexnější algoritmy. Pro tyto účely jsou definovány specializované konstrukce výrazů. Jejich přehled následuje v tabulce 10.2.

Struktura výrazu	Popis
$\frac{pravda \quad nepravda}{podmínka}$	podmíněný výraz
$\begin{pmatrix} výraz\ 1 \\ výraz\ 2 \\ \vdots \\ výraz\ n \end{pmatrix}$	sekvence výrazů
$\left\{\frac{tělo\ cyklu}{podmínka}\right\}$	cyklus (tělo cyklu, kterým může být jeden výraz nebo sekvence výrazů, se vykonává pouze za předpokladu splnění podmínky)

Tabulka 10.2. Specializované konstrukce

S využitím výše zmíněných konstrukcí lze jako lambda výrazy symbolicky zapsat běžné algoritmy. Jako příklad takového zápisu je v podobě lambda výrazu níže uveden algoritmus určení výsledku funkce *signum*:

$$signum = \left(\lambda x \left| \frac{0 \quad \frac{1 \quad 0}{x > 0}}{x = 0} \right. \right)$$

10.2 Analogie obou domén

Na základě rozboru informací z úvodní rešeršní části práce, byly identifikovány základní vlastnosti jak objektového, tak servisního paradigmatu, u nichž lze způsobem ontologické analogie nalézt shodu. Tabulka 10.3 uvádí všechny vybrané prvky servisního paradigmatu a jim analogické prvky paradigmatu objektového.

Je nutno podotknout, že pro obě skupiny vybraných prvků platí, že tvoří pouze podmnožinu všech vlastností daného paradigmatu.

Vlastnosti vyznačené v tabulce 10.3 kurzívou byly vybrány jako klíčové pro úspěšnou aplikaci vybrané techniky objektového modelování při návrhu servisně orientovaného systému a jsou proto formálně popsány v následujících kapitolách.

SOA	Objektový model
služba	*třída*
operace	*metoda*
funkční část kontraktu služby	*protokol třídy*
abstrakce	zapouzdření
znovu použitelnost	znovu použitelnost
složitelnost	skládání

Tabulka 10.3. Analogické prvky obou domén

10.3 Objektově orientované paradigma

Základní vlastnosti objektově orientovaného paradigmatu byly představeny v kapitole 6. Vybrané klíčové prvky identifikované při ontologickém rozboru problémové domény jsou podrobněji popsány a formálně definovány v této části práce. Formální popis vybraných prvků objektového modelu je založen na definicích uvedených v (Merunka, 2008).

Def. 10.1 Třída

V objektově orientovaném paradigmatu pojmem třída označujeme aparát umožňující vytvořit společný popis množiny dat a množiny metod daného druhu objektů. Platí:

$\forall C \in \mathbb{C}$, kde $\mathbb{C}$ je universem všech tříd.

Třídu každého objektu lze zapsat jako $class(o)$, kde o představuje objekt náležející dané třídě. Třída svým objektům poskytuje definice chování, tj. metod, které objekty mohou aplikovat na svoje data.

Objekt náležející třídě je označován jako instance třídy. Pokud je definována funkce *inst*, která dává množinu všech instancí třídy a dále je definováno universum všech přípustných objektů $\mathbb{O}$, pak platí:

$$o \in \mathbb{O}, C \in \mathbb{C}: \quad C = class(o) \leftrightarrow o \in inst(C)$$

Pokud tedy objekt o náleží třídě C, je její instancí. Opačně, pokud je objekt o instancí třídy C, pak jí náleží.

Def. 10.2 Data

Data (datové objekty) v kontextu analogie k servisním službám představují v objektovém paradigmatu všechny instance dané třídy obsahující konkrétní data:

$$\forall o: o \in inst(C)$$

Objekty pro operace s těmito daty mohou využívat metod definovaných svou třídou.

Def. 10.3 Metoda

Metoda v kontextu objektově orientovaného paradigmatu představuje operaci, která je definována třídou a lze jí vyvolat pomocí předání zprávy objektu (viz def. 9.6).

Metoda je definována jako uspořádaná dvojice:

$$m = \langle h, \Lambda \rangle,$$

kde h představuje hlavičku metody a Λ je lambda výraz popisující co metoda provádí. Hlavička metody slouží ke svázání konkrétní metody se zprávou přijímanou daným datovým objektem.

Def. 10.4 Množina všech metod třídy

Funkce *meth* pro danou třídu dává množinu všech metod, které jsou třídou definovány. Platí pak:

$$meth(C) = (m_1, m_2, m_3, \cdots, m_n),$$

kde $m_i = \langle h, \Lambda \rangle$.

Def. 10.5 Rozhraní třídy

Rozhraním třídy rozumíme množinu všech zpráv, které lze instancím dané třídy zaslat. Formálně ji lze definovat:

$$\Pi(C) = (\mu_1, \mu_2, \mu_3, \cdots, \mu_n),$$

kde $C \in \mathbb{C}$ je daná třída a $\mu_1, \cdots, \mu_n$ jsou jednotlivé zprávy, které jsou svázány s metodami, které třída definuje, a jejich požadovanými parametry. Pro každou zprávu tedy platí:

$$\mu = (h, \{p_i\}),$$

kde h je hlavička patřičné metody a $\{p_i\}$ je množina požadovaných parametrů, přičemž:

$$\forall p_i \in \{p_i\}: p_i \in \mathbb{O}$$

Def. 10.6 Volání metody

Operace volání metody (method look-up) je definována jako:

$$o \lhd \mu = \frac{\Lambda(\{p_i\}) \qquad\qquad \emptyset}{\mu \in \Pi\big(class(o)\big) \ \wedge \ \mu = (h, \{p_i\}) \ \wedge \ \langle h, \Lambda \rangle \in meth\big(class(o)\big)}$$

Pro úspěšné spárování metody s obdrženou zprávou a její následné vykonání je nutné, aby byly splněny podmínky uvedené ve výrazu výše. Tedy:

- zpráva musí být součástí protokolu třídy, jíž objekt náleží;
- zpráva sestává z identifikátoru volané metody a množiny parametrů, které mají být metodě předány;

- identifikátor metody musí být součástí definice metody, která je prvkem množiny metod třídy.

10.4 Servisně orientované paradigma

Základní vlastnosti servisně orientovaného paradigmatu jsou popsány v kapitole 7. Vybrané klíčové vlastnosti analogické k vlastnostem objektového modelu budou formálně popsány v této kapitole.

Def. 10.7 Služba

Službou v kontextu servisně orientovaného paradigmatu rozumíme softwarovou komponentu realizující definované operace nad danými daty. Platí:

$$\forall S \in \mathbb{S}, \text{kde } \mathbb{S} \text{ je universem všech služeb.}$$

Def. 10.8 Funkční část kontraktu služby

Služba obsahuje definice operací, které může provádět a definici vyžadovaných vstupních parametrů pro každou operaci. Informace o voláních operací a jejich parametrech jsou součástí funkční částí kontraktu služby:

$$\Phi(S) = (\varphi_1, \varphi_2, \varphi_3, \cdots, \varphi_n),$$

kde $S \in \mathbb{S}$ je daná služba a $\varphi_1, \varphi_2, \cdots, \varphi_n$ jsou volání jednotlivých operací této služby, pro která platí:

$$\varphi = (h, \{p_i\}).$$

Def. 10.9 Data

Pro $\{p_i\}$ platí, že $\{p_i\} \subset \mathbb{D}$, kde $\mathbb{D}$ je množina všech využitelných dat. Zároveň platí, že výsledek $S \lhd (h, \{p_i\}) \subset \mathbb{D}$.

Výsledek volání operace je tedy taktéž podmnožinou množiny všech využitelných dat $\mathbb{D}$. Výsledek jedné operace tedy lze předat jako parametr volání další operace.

Pro situaci, kdy je výsledek volání operace $S_1 \lhd \varphi_1$ využit jako parametr operace $S_2 \lhd \varphi_2$, potom platí:

$$S_1 \lhd \varphi_1 \xRightarrow[substituce]{} S_1 \lhd (h_1, \{p_i\}) \xRightarrow[redukce]{} \Lambda_1 \{p_i\}$$

$$S_2 \lhd \varphi_2 \xRightarrow[substituce]{} S_2 \lhd \big(h_2, (\Lambda_1 \{p_i\})\big) \xRightarrow[redukce]{} \Lambda_2 (\Lambda_1 \{p_i\})$$

Def. 10.10 Operace

Operace služby představuje uzavřený blok funkcionality služby, který provádí definovanou činnost transformující vstupní data na data výstupní.

Operace služby je definována jako uspořádaná dvojice:

$$o = \langle h, \Lambda \rangle,$$

kde h je hlavička operace a Λ je lambda výraz popisující činnost prováděnou operací.

Def. 10.11 Množina operací služby

Funkce *op* pro danou službu dává množinu všech operací, které jsou touto službou definovány. Platí tedy:

$$op(S) = (o_1, o_2, o_3, \cdots, o_n),$$

kde $o_i = \langle h, \Lambda \rangle$.

Def. 10.12 Volání operace

Volání operace služby je definováno jako:

$$S \lhd \varphi = \frac{\Lambda(\{p_i\}) \qquad\qquad \emptyset}{\varphi \in \Phi(S) \ \wedge \ \varphi = (h, \{p_i\}) \ \wedge \ \langle h, \Lambda \rangle \in op(S)}$$

Pro úspěšné vykonání operace musí být splněny vyjádřené podmínky. Tedy:

- volání operace je součástí funkční části kontraktu služby;
- volání operace obsahuje platné parametry;
- volání operace lze spárovat s patřičnou definicí operace, která je součástí množiny operací služby.

11 Využití objektových návrhových vzorů v oblasti modelování servisně orientovaných systémů

Nalezená analogie a možnost popsat vybrané vlastnosti servisně orientovaného paradigmatu s pomocí objektového kalkulu, jak je provedeno v kapitolách 10.3 a 10.4, připravují půdu pro aplikaci vybrané techniky objektového modelování při návrhu servisně orientovaného systému.

Návrh takových systémů je obvykle spojován s procesním modelováním a orchestrací služeb v souvislosti s procesy těchto modelů. Jak však ukazuje i struktura modelů v rámci SOMF, je důležité věnovat se i modelování struktury služeb, jejich vazeb a komunikací (Bell, 2008) (Bell, 2010).

Jako vhodná technika byly vybrány návrhové vzory, které jsou v oblasti objektového modelování a programování osvědčeným prostředkem pro řešení řady problémů. Cílem aplikace vybrané skupiny návrhových vzorů v oblasti návrhu servisně orientovaných systémů je nabídnout analytikům a vývojářům takových systémů nástroj, se kterým jsou již seznámeni a využívají ho v jiné oblasti. Cílovou skupinou tak nejsou konzumenti finálních služeb, ale ti, kteří se na tvorbě těchto služeb podílejí.

Vhodnou oblastí pro využití návrhových vzorů může být například integrace již existujících služeb (tzv. „legacy services“) do nové struktury, kdy mnohdy nejde díky jejich povaze či využité technologii postupovat přesně podle pravidel pro návrh čistě servisních systémů (Bell, 2008).

Z rozsáhlé knihovny návrhových vzorů, která byla popsána v kapitole 8, byly pro ověření vybrány následující návrhové vzory:

- Adaptér,
- Proxy,
- Fasáda.

Tyto vzory byly zvoleny proto, že je lze do servisně orientovaného modelu převést bez nutnosti definice dalších operací nad rámec operací definovaných pro objektový model v kapitole 10.3, respektive v kapitole 10.4 pro model servisně orientovaný. Neobsahují tedy například skládání ani dědění.

Využití dalších návrhových vzorů a jejich případná kombinace s jinými technikami převzatými z oblasti objektového modelování, či s technikami nativními pro oblast servisně orientovaného návrhu, se nabízí jako potenciální oblast pro další vědeckou práci.

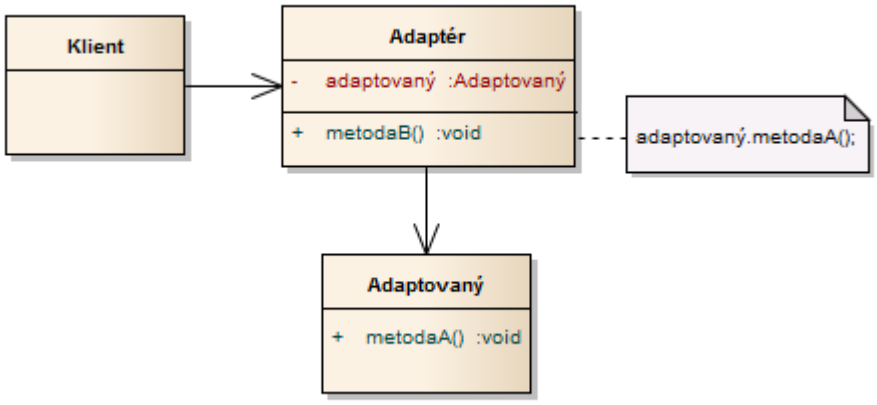
11.1 Struktura zpracovaných návrhových vzorů

Popis představovaných návrhových vzorů má strukturu podobnou definicím návrhových vzorů pro objektové modelování. Dokumentace každého návrhového vzoru zpracovaného pro servisně orientovaný model obsahuje:

- název návrhového vzoru,
- popis problému, který vzor řeší,
- popis řešení,
- možné oblasti využití,
- strukturu vzoru v notaci SOMF pro analytický a logický model,
- seznam participantů.

Pro potřeby této práce je dokumentace každého vzoru doplněna o popis a UML třídní diagram původní podoby vzoru pro objektové modelování, formální popis vykonávání každého zpracovaného návrhového vzoru a příklad vykonávání s konkrétními hodnotami parametrů. Průběh vykonávání vzoru je znázorněn i pomocí diagramu transakčních aktivit v notaci SOMF.

11.2 Návrhový vzor Adaptér



Obr. 11.1. Struktura návrhového vzoru Adaptér v objektovém modelu.

Návrhový vzor Adaptér v objektovém modelování řeší problém konverze rozhraní jednoho objektu na rozhraní jiného objektu. Při jeho implementaci se obvykle využívá dědění, protože většina programovacích jazyků je striktně typová. V čistě objektovém jazyku, jakým je například Smalltalk, lze tento vzor implementovat i bez použití dědění (Merunka, 2008). Struktura takto implementovaného vzoru je zobrazena na obrázku 11.1.

11.2.1 Popis vzoru pro servisně orientovaný model

Název vzoru: Adaptér

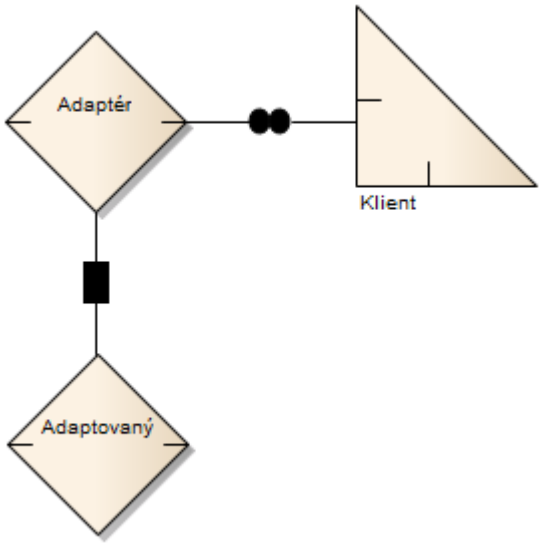
Popis řešeného problému: Definice volání operací jedné služby je potřebné přeložit do definic volání jiné služby.

Popis řešení: Mezi klienta a službu je vložena další služba sloužící jako překladač z jedné definice do druhé.

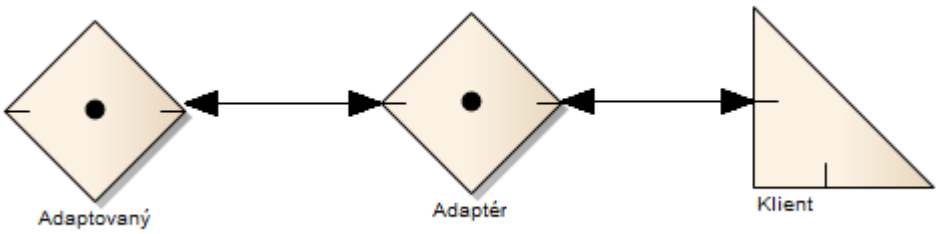
Oblasti využití: Kdekoliv je nutné upravit definici volání služby, např. v situaci, kdy je nutné již existující neupravovatelnou službu začlenit do systému, kde je vyžadována konformita s určitým standardem definic operací.

Participanti: Adaptovaný (služba, jejíž definice je překládána), adaptér (služba, která definici překládá), klient (využívá adaptovanou službu pomocí adaptéru).

Struktura vzoru pro analytický model:



Struktura vzoru pro logický model:



11.2.2 Formální definice pro servisně orientovaný model

Nechť $s \in \mathbb{X}$ a platí:

$$\varphi_s \in \Phi(s),$$

$$o_s = \langle h_s, \Lambda_s \rangle \ \wedge \ (h_s, \{p_i\}) = \varphi_s \ \wedge \ o_s \in op(s).$$

Dále nechť $a \in \mathbb{X}$ a platí:

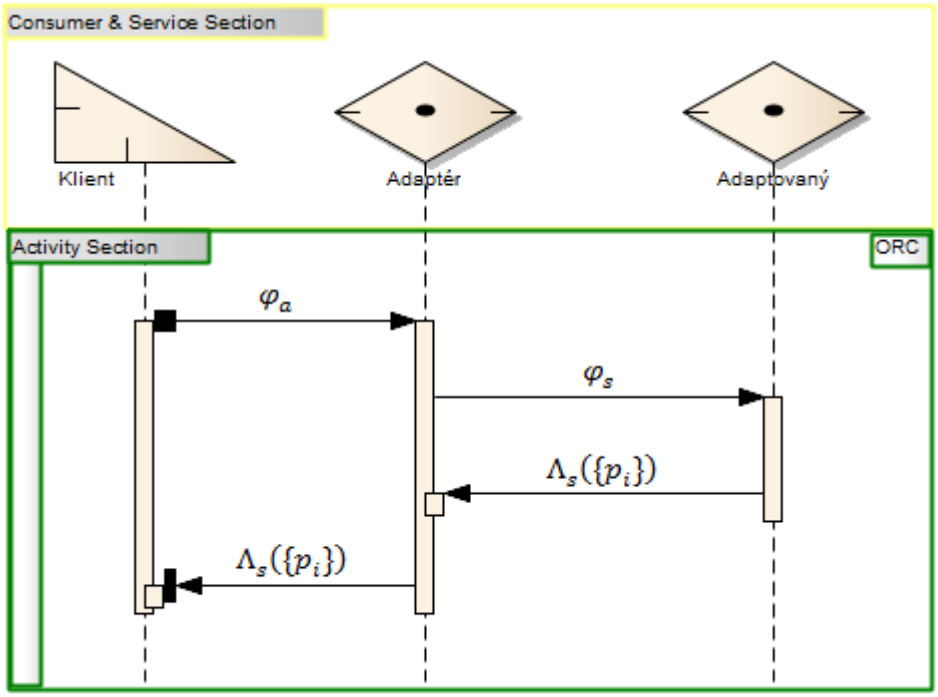
$$\varphi_a \in \Phi(a),$$

$$o_a = \langle h_a, \Lambda_a \rangle \ \wedge \ (h_a, \{p_i\}) = \varphi_a \ \wedge \ o_a \in op(a) \ \wedge \ \Lambda_a = (s \triangleleft \varphi_s).$$

Průběh vykonávání vzoru pak zapíšeme takto:

$$a \triangleleft \varphi_a \xrightarrow[substituce]{} \Lambda_a(\{p_i\}) \xrightarrow[redukce]{} s \triangleleft \varphi_s \xrightarrow[substituce]{} \Lambda_s(\{p_i\})$$

Následující obrázek nabízí v podobě diagramu transakčních aktivit SOMF zobrazení průběhu vykonávání vzoru Adaptér v obecné formě. Diagram transakčních aktivit je podrobně popsán v (Methodologies Corporation, 2011b).



Obr. 11.2. Znázornění průběhu vzoru Adaptér pomocí diagramu transakčních aktivit SOMF.

Následující příklad ilustruje aplikaci adaptéru *mAdapter* s operací *naDruhou* na konkrétní službu *math* a její operaci *square*.

Nechť:

$$math, mAdapter \in \mathbb{X},$$

$$square \in \Phi(math),$$

$$naDruhou \in \Phi(mAdapter),$$

$$\Lambda_{square} = (\lambda p|\, p^2),$$

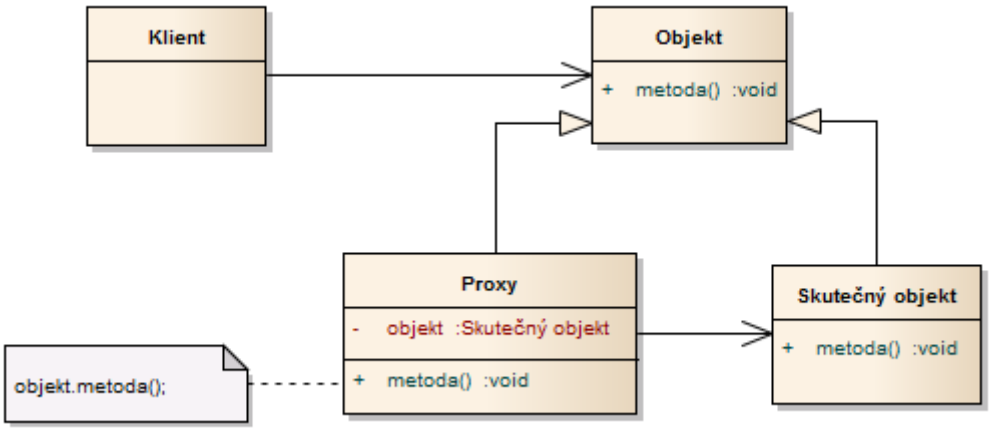
$$\Lambda_{naDruhou} = \left(\lambda p\middle|\, math \lhd square(p)\right).$$

Pro konkrétní hodnotu parametru p=3 pak průběh vykonávání zapíšeme jako následující sekvenci formálních operací:

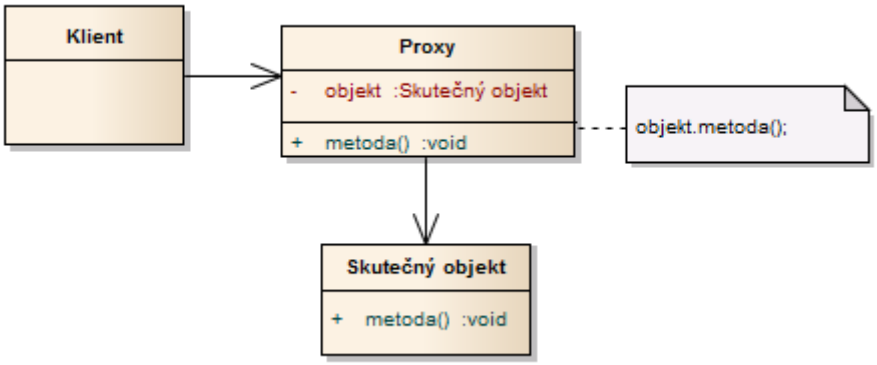
$$mAdapter \lhd naDruhou(3) \underset{substituce}{\Longrightarrow} \Lambda_{naDruhou}(3)$$
$$\underset{redukce}{\Longrightarrow} math \lhd square(3) \underset{substituce}{\Longrightarrow} \Lambda_{square}(3) \underset{redukce}{\Longrightarrow} 9$$

11.3 Proxy

Návrhový vzor Proxy v objektovém modelu slouží k řízení přístupu k jinému objektu. Vzor Proxy ve většině programovacích jazyků využívá dědění (viz obr. 11.3). Podobně jako návrhový vzor Adaptér ho však lze na objektové úrovni realizovat bez nutnosti dědění využívat (obr. 11.4).



Obr. 11.3. Struktura vzoru Proxy v objektovém modelu (s využitím dědění)



Obr. 11.4. Struktura vzoru Proxy v objektovém modelu (verze bez dědění)

11.3.1 Popis vzoru pro servisně orientovaný model

Název vzoru: Proxy

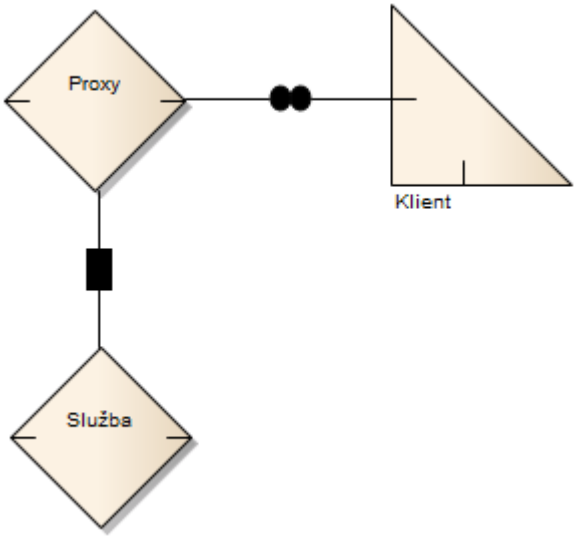
Popis řešeného problému: Je potřebné odstínit službu od klienta.

Popis řešení: Mezi klienta a službu je vložena další služba sloužící jako zástupce odstíněné služby.

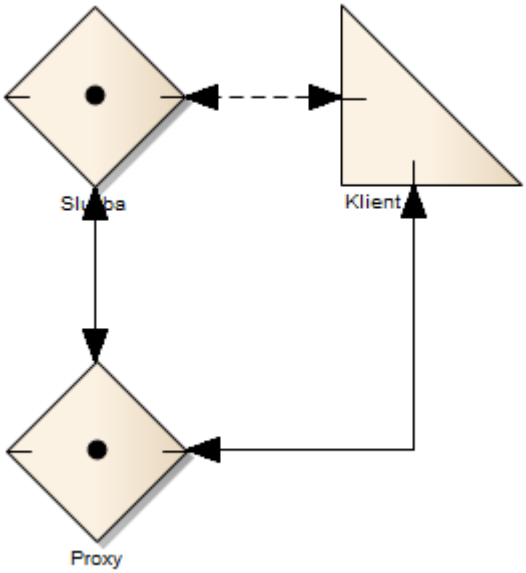
Oblasti využití: Kdekoliv je nutné odstínit službu od klienta. Např. pokud by měla být služba využívána z méně bezpečného prostředí a je nutné zabezpečit přístup k ní (bezpečnostní proxy).

Participanti: Odstíněná služba (služba, kterou je potřeba odstínit od klienta), proxy (služba, která zabezpečuje odstínění), klient (přistupuje k odstíněné službě pomocí proxy).

Struktura pro analytický model:



Struktura pro logický model:



11.3.2 Formální definice pro servisně orientovaný model

Nechť $s, p \in \mathbb{X}$ a platí:

$$\varphi \in \Phi(s),$$

$$\Phi(s) = \Phi(p).$$

Dále nechť:

$$o_s \in op(s) \ \wedge \ o_s = \langle h_s, \Lambda_s \rangle \ \wedge \ (h_s, \{p_i\}) = \varphi,$$

$$o_p \in op(p) \ \wedge \ o_p = \langle h_p, \Lambda_p \rangle \ \wedge \ \left(h_p, \{p_i\}\right) = \varphi$$

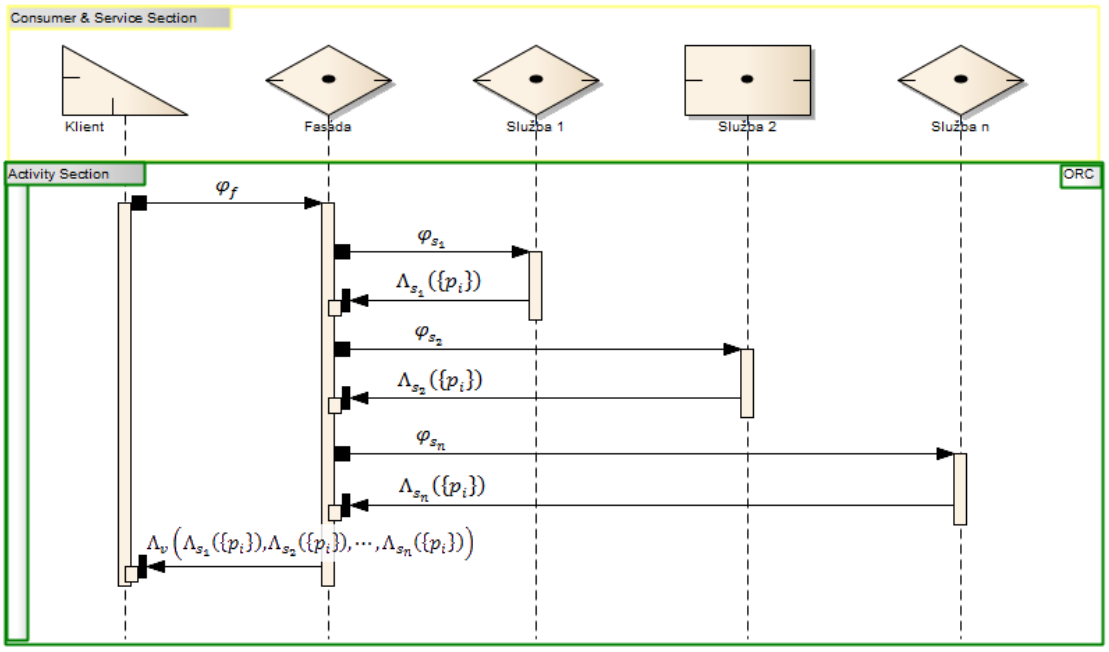
a platí:

$$\Lambda_s \neq \Lambda_p,$$

$$\Lambda_p = (s \triangleleft \varphi).$$

Průběh vykonávání vzoru pak zapíšeme takto:

$$p \triangleleft \varphi \underset{volání}{\Longrightarrow} \Lambda_p(\{p_i\}) \underset{redukce}{\Longrightarrow} s \triangleleft \varphi \underset{volání}{\Longrightarrow} \Lambda_s(\{p_i\})$$



Obr. 11.5. Diagram transakčních aktivit SOMF zobrazující vykonávání vzoru Proxy.

Průběh vykonávání vzoru Proxy v obecné formě je vyobrazen na obrázku 11.5.

Následující příklad ilustruje aplikaci vzoru proxy v podobě služby *proxyMath* na konkrétní službu *math* a její operaci *square*.

Nechť:

$$math, proxyMath \in \mathbb{X},$$

$$square \in \Phi(math),$$

$$\Phi(math) = \Phi(proxyMath),$$

$$\Lambda_{square} = (\lambda p |\, p^2),$$

$$\Lambda_{proxy} = \left(\lambda d \middle|\, math \lhd square(d)\right),$$

$$o_s \in op(math) \wedge o_s = \langle h_s, \Lambda_{square} \rangle \wedge (h_s, \{p_i\}) = square,$$

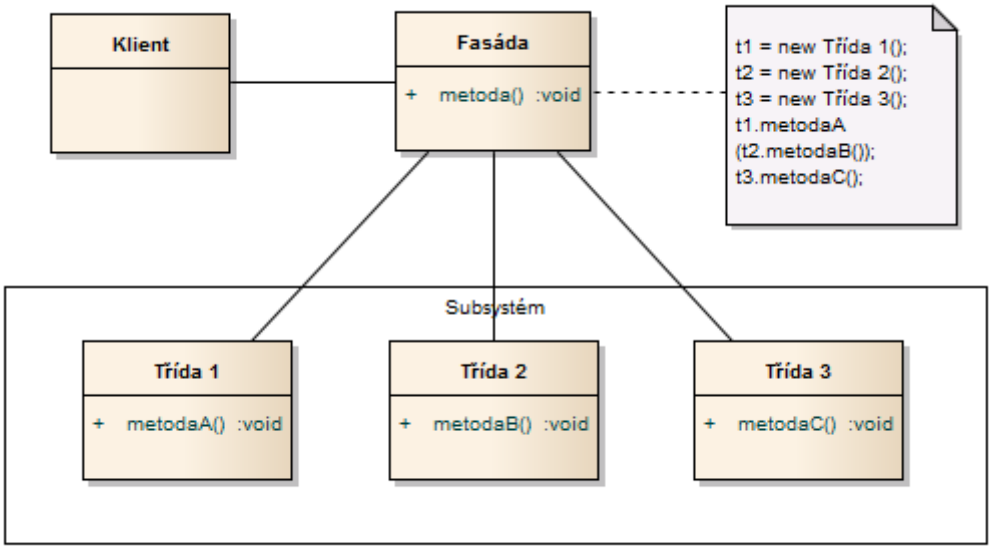
$$o_p \in op(math) \wedge o_p = \langle h_p, \Lambda_{proxy} \rangle \wedge \left(h_p, \{p_i\}\right) = square.$$

Pro konkrétní hodnotu parametru p=3 pak průběh vykonávání zapíšeme jako následující sekvenci operací:

$$proxyMath \lhd square(3) \underset{volání}{\Longrightarrow} \Lambda_{proxy}(3)$$
$$\underset{redukce}{\Longrightarrow} math \lhd square(3) \underset{volání}{\Longrightarrow} \Lambda_{square}(3) \underset{redukce}{\Longrightarrow} 9$$

11.4 Fasáda

Návrhový vzor Fasáda v objektovém modelování představuje prostředek umožňující zjednodušení přístupu k subsystému. Struktura návrhového vzoru Fasáda je zobrazena na obr. 11.6.



Obr. 11.6. Struktura návrhového vzoru Fasáda v objektovém modelu.

11.4.1 Popis vzoru pro servisně orientovaný model

Název vzoru: Fasáda

Popis řešeného problému: Je potřebné zjednodušit volání operací subsystému představovaného množinou služeb.

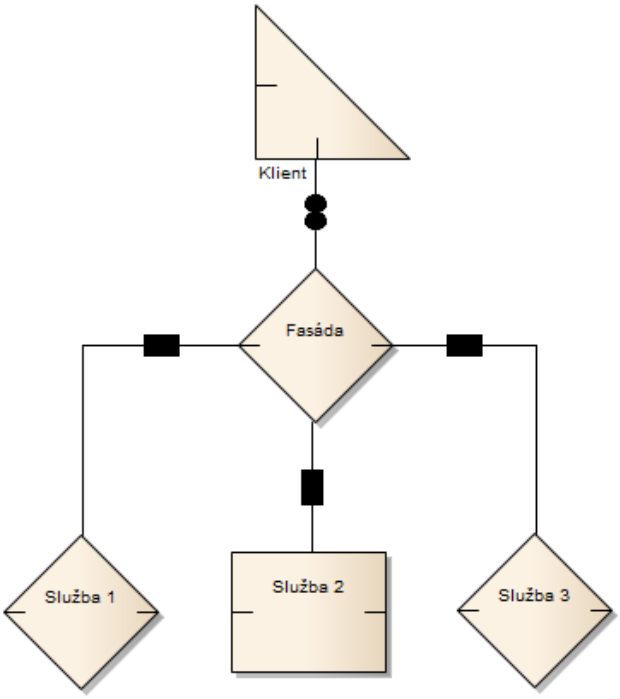
Popis řešení: Mezi klienta a množinu využívaných služeb je vložena další služba, která odstiňuje subsystém od klienta a zjednodušuje volání operací subsystému tím, že vytvoří jeden vstupní bod, se kterým klient komunikuje.

Oblasti využití: Kdekoliv je nutné zjednodušit volání operace při zachování stávající struktury služeb, které jednotlivé operace vykonávají.

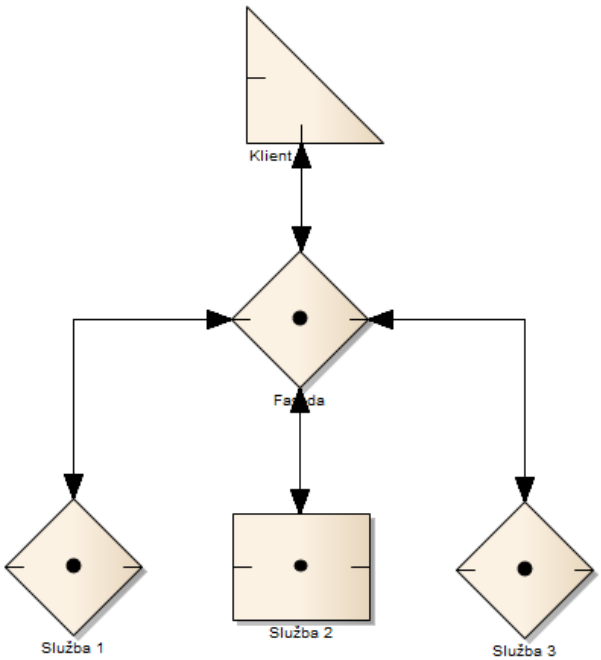
Participanti: Služby subsystému (služby vykonávající v rámci subsystému dílčí operace), fasáda (služba, která místo klienta komunikuje se

subsystémem), klient (využívá odstíněný subsystém pomocí vstupního bodu tvořeného službou fasády).

Struktura pro analytický model:



Struktura pro logický model:



11.4.2 Formální definice pro servisně orientovaný model

Nechť $s_1, s_2, \cdots, s_n \in \mathbb{S}, \mathbb{S} \subset \mathbb{X}, f \in \mathbb{X}$ a platí:

$$\varphi_f \in \Phi(f) \ \wedge \ o_f \in op(f) \ \wedge \ o_f = \langle h_f, \Lambda_f \rangle \ \wedge \ \left(h_f, \{p_i\}\right) = \varphi_f,$$

$$\forall s_i \in \mathbb{S}: \varphi_{s_i} \in \Phi(s_i) \ \wedge \ o_{s_i} \in op(s_i) \ \wedge \ o_{s_i} = \langle h_{s_i}, \Lambda_{s_i} \rangle \ \wedge \ \left(h_{s_i}, \{p_i\}\right) = \varphi_{s_i}.$$

Dále nechť:

$$\Lambda_f = \left(\lambda p \middle| \Lambda_v \begin{pmatrix} v_1 \Leftarrow s_1 \lhd \varphi_{s_1} \\ v_2 \Leftarrow s_2 \lhd \varphi_{s_2} \\ \vdots \\ v_n \Leftarrow s_n \lhd \varphi_{s_n} \end{pmatrix} \right),$$

kde $v_1, v_2, \cdots, v_n \in \mathbb{D}$ jsou výsledky jednotlivých operací a funkce Λ_v představuje mechanismus tyto výsledky agregující.

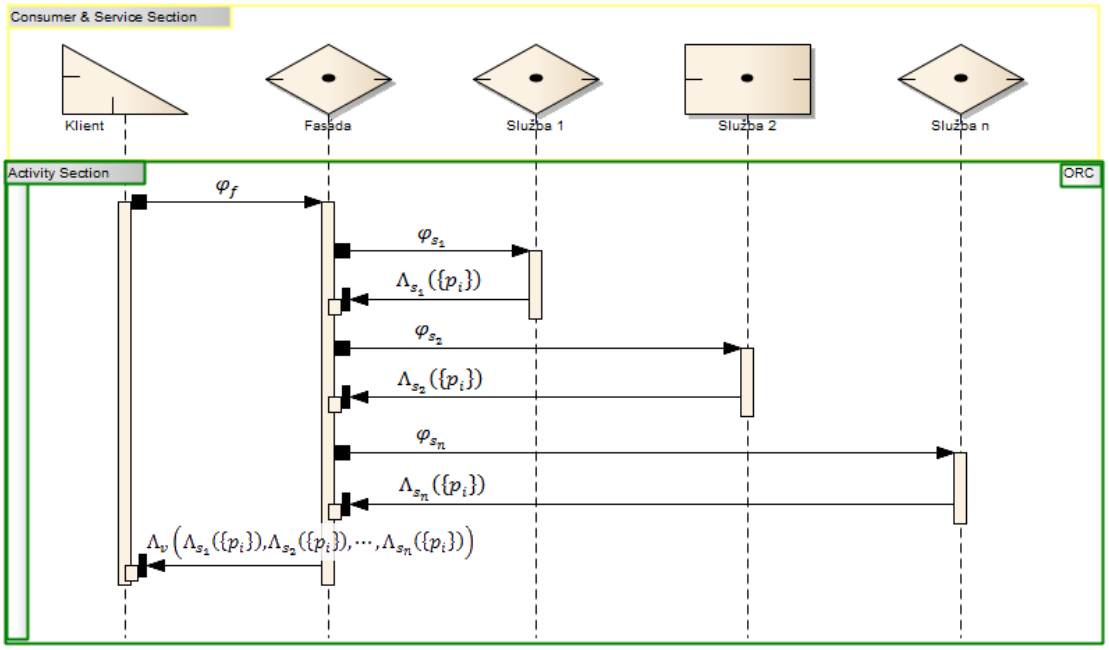
Průběh vykonávání vzoru pak zapíšeme takto:

$$f \lhd \varphi \underset{substituce}{\Longrightarrow} \Lambda_f(\{p_i\})$$

$$\underset{redukce}{\Longrightarrow} \Lambda_v \begin{pmatrix} v_1 \Leftarrow s_1 \lhd \varphi_{s_1} \\ v_2 \Leftarrow s_2 \lhd \varphi_{s_2} \\ \vdots \\ v_n \Leftarrow s_n \lhd \varphi_{s_n} \end{pmatrix} \underset{substituce}{\Longrightarrow} \Lambda_v \begin{pmatrix} v_1 \Leftarrow \Lambda_{s_1}(\{p_i\}) \\ v_2 \Leftarrow \Lambda_{s_2}(\{p_i\}) \\ \vdots \\ v_n \Leftarrow \Lambda_{s_n}(\{p_i\}) \end{pmatrix}$$

$$\underset{redukce}{\Longrightarrow} \Lambda_v(v_1, v_2, \cdots, v_n)$$

Průběh vykonávání vzoru v podobě diagramu transakčních aktivit SOMF je zobrazen na obrázku 11.7.



Obr. 11.7. Průběh vzoru Fasáda jako diagram transakčních aktivit SOMF

Následující příklad demonstruje využití vzoru Fasáda v podobě služby *geography* využívající operace služby *math*. Služba *geography* poskytuje operaci *distance*, která vypočte vzdálenost na velké kružnici dvou míst definovaných parametry v podobě jejich zeměpisných souřadnic. Využívaná služba *math* poskytuje matematické operace pro provedení tohoto výpočtu.

Nechť:

$$math, geography \in \mathbb{X},$$

$$acos, cos, sin \in \Phi(math),$$

$$distance \in \Phi(geography),$$

$$\Lambda_{arccos} = \left(\lambda p \middle| \arccos(p)\right),$$

$$\Lambda_{cos} = \left(\lambda p \middle| \cos(p)\right),$$

$$\Lambda_{sin} = \left(\lambda p \middle| \sin(p)\right),$$

$$\Lambda_f = \left(\lambda lt_1 \lambda ln_1 \lambda lt_2 \lambda ln_2 \middle| \Lambda_v \begin{pmatrix} v_1 \Leftarrow math \lhd sin(lt_1) \\ v_2 \Leftarrow math \lhd sin(lt_2) \\ v_3 \Leftarrow math \lhd cos(lt_1) \\ v_4 \Leftarrow math \lhd cos(lt_2) \\ v_5 \Leftarrow math \lhd cos(ln_1 - ln_2) \end{pmatrix} \right),$$

$$\Lambda_v = \left(\lambda v_1 \lambda v_2 \lambda v_3 \lambda v_4 \lambda v_5 \middle| math \lhd arccos(v_1 \cdot v_2 + v_3 \cdot v_4 \cdot v_5) \right).$$

Průběh vykonávání vzoru pro tento příklad v obecné podobě (bez zadání konkrétních hodnot zeměpisné šířky a délky) zapíšeme takto:

$$geography \lhd distance \underset{substituce}{\Longrightarrow} \Lambda_f(lt_1, ln_1, lt_2, ln_2)$$

$$\underset{redukce}{\Longrightarrow} \Lambda_v \begin{pmatrix} v_1 \Leftarrow math \lhd sin(lt_1) \\ v_2 \Leftarrow math \lhd sin(lt_2) \\ v_3 \Leftarrow math \lhd cos(lt_1) \\ v_4 \Leftarrow math \lhd cos(lt_2) \\ v_5 \Leftarrow math \lhd cos(ln_1 - ln_2) \end{pmatrix}$$

$$\underset{substituce}{\Longrightarrow} \Lambda_v \begin{pmatrix} v_1 \Leftarrow \Lambda_{sin}(lt_1) \\ v_2 \Leftarrow \Lambda_{sin}(lt_2) \\ v_3 \Leftarrow \Lambda_{cos}(lt_1) \\ v_4 \Leftarrow \Lambda_{cos}(lt_2) \\ v_5 \Leftarrow \Lambda_{cos}(ln_1 - ln_2) \end{pmatrix}$$

$$\underset{redukce}{\Longrightarrow} \Lambda_v(v_1, v_2, v_3, v_4, v_5)$$

$$\underset{redukce}{\Longrightarrow} math \lhd arccos(v_1 \cdot v_2 + v_3 \cdot v_4 \cdot v_5)$$

$$\underset{substituce}{\Longrightarrow} \Lambda_{arccos}(v_1 \cdot v_2 + v_3 \cdot v_4 \cdot v_5)$$

12 Diskuse

Nalezená podobnost mezi objektovým a servisním modelem, která byla popsána v kapitole 10, umožnila pro vybrané návrhové vzory z oblasti objektového modelování vytvoření definic pro oblast servisně orientovaného paradigmatu a jejich formální popis pomocí prostředků objektového kalkulu. Otevírá se tak možnost aplikovat na určité oblasti servisního modelu prostředky využívané při modelování objektovém.

Jako prostor pro další práci se nabízí hledání dalších prvků servisně orientovaného paradigmatu, které by byly analogické k prvkům paradigmatu objektového, což by umožnilo například aplikaci dalších objektových návrhových vzorů, které za stávající situace využít nelze.

V oblasti SOA zároveň probíhá výzkum i v jiných oblastech, například normalizace operací služeb, či jejich granularity. U tématu vzájemného vztahu granularity služeb a parametrů jejich rozhraní (nastíněno v (Feuerlicht, 2006)) se nabízí například otázka, zda by díky nalezené shodě na tuto problematiku nebylo možné aplikovat některou z metod objektové (datové) normalizace.

13 Závěr

Tato práce byla zaměřena na problematiku návrhu distribuovaných informačních systémů se zaměřením na servisně orientovanou architekturu, konkrétně na možnosti využití metod objektového modelování při návrhu servisně orientovaných systémů.

Hlavním cílem práce bylo navrhnout možnost aplikace vybrané techniky objektového modelování při řešení problému v oblasti modelování servisně orientovaných systémů.

Pro splnění tohoto cíle provedeny následující dílčí kroky.

1. Byly analyzovány literární zdroje týkající se problematiky distribuovaných objektových systémů a servisně orientované architektury. Byly popsány základní technologie a modelovací techniky využitelné v obou oblastech.
2. Na základě analýzy literatury z bodu 1 byly identifikovány klíčové vlastnosti obou paradigmat, mezi kterými pak byly pomocí ontologické analogie nalezeny vzájemně analogické prvky.
3. Vybrané analogické prvky byly formálně popsány a definovány s využitím objektového kalkulu.
4. S využitím výše zmíněných formálních definic byly pro servisně orientovaný model definovány tři vybrané objektové návrhové vzory, které byly formálně popsány a byla navržena i jejich podoba v analytickém a logickém modelu servisně orientovaného modelovacího frameworku SOMF.

Aplikace objektového kalkulu na oblast servisně orientovaného modelování otevírá možnosti návazného výzkumu například v těchto oblastech:

- hledání analogie s dalších prvků objektového modelu s prvky modelu servisního,

- využití dalších návrhových vzorů a prostředků objektového modelování při tvorbě analytického modelu SOA,
- aplikace metod objektové normalizace na množiny vstupních parametrů a jejich vliv na granularitu služeb.

Představuje tak východisko pro další vědeckou práci rozvíjející toto téma.

Literární zdroje

Armstrong, Deborah J. 2006. The Quarks of Object Oriented Development. *Communications of the ACM.* 2006, Sv. 49, 2.

Barry, Douglas K. 2003. Prior Service-Oriented Architecture specifications. *Web Services and Service-Oriented Architectures.* [Online] 2003. [Citace: 2. 9 2010.] http://www.service-architecture.com/web-services/articles/prior_service-oriented_architecture_specifications.html.

—. **2003.** Service-oriented architecture (SOA) definition. *Web Services and Service-Oriented Architectures.* [Online] 2003. [Citace: 23. 8 2010.] http://www.service-architecture.com/web-services/articles/service-oriented_architecture_soa_definition.html.

Bell, Michael. 2008. *Service-Oriented Modeling (SOA): Service Analysis, Design, and Architecture.* Hoboken : John Wiley & Sons, 2008. ISBN 978-0-470-14111-3.

—. **2010.** *SOA Modeling Patterns for Service-Oriented Discovery and Analysis.* Hoboken : John Willey and Sons, 2010. ISBN 978-0-470-48197-4.

BPMI. 2008. *Business Process Management Initiative.* [Online] BPMI, 2008. [Citace: 15. 9 2010.] http://www.bpmi.org.

Brožek, Jiří. 2008. Formal Form Of Method Invocation And Differences Of The Distributed Approach. *Proceeding Of The Doctoral Conference Thing Together 2008.* 2008.

Caromel, Denis a Henrio, Ludovic. 2005. *A Theory of Distributed Objects.* Heidelberg : Springer Verlag, 2005. ISBN 978-3-540-20866-2.

Clements, Paul et. al. 2009. *Documenting Software Architectures: Views and beyond.* Boston : Addison-Wesley, 2009.

Craft.CASE ltd. Craft.CASE. [Online] Craft.CASE ltd. http://www.craftcase.com.

Eeles, Peter. 2006. What is a software architecture? *IBM Technical Library*. [Online] 15. 2 2006. [Citace: 12. 5 2010.] http://www.ibm.com/developerworks/rational/library/feb06/eeles.

Emmerich, Wolfgang. 2000. *Engineering Distributed Objects*. Hoboken : John Wiley & Sons, 2000. ISBN 978-0471986577.

Erl, Thomas. 2006. *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River : Prentice Hall, 2006. ISBN 978-0-13-185858-9.

—. 2007. *SOA Principles of Service Design*. Upper Saddle River : Prentice Hall, 2007. ISBN 978-0132344821.

Feuerlicht, George. 2006. Service granularity considerations based on data properties of interface parameters. *COMPUTER SYSTEMS SCIENCE AND ENGINEERING*. 2006, Sv. 21, 4.

Gamma, Erich, a další. 1994. *Design Patterns: Elements of Object-Oriented Software Architecture*. Reading : Addison-Wesley, 1994. ISBN 0-201-63361-2.

Garlan, David a Shaw, Mary. 1993. An Introduction to Software Architecture. [autor knihy] V. Ambriola a G. Tortora. *Advances in Software Engineering, Volume I*. New Jersey : World Scientific Publishing Company, 1993.

—. 1996. *Software Architecture: Perspectives on an Emerging Discipline*. Upper Saddle River : Prentice Hall, 1996.

IEEE 1471-2000. IEEE Recommended Practice for Architectural Description for Software-Intensive Systems.

Knott, R. P., Merunka, V. a Polák, J. 2006. Part 15: BORM Methodology. [autor knihy] Liping Liu a Borislav Roussev. *Management of the Object-Oriented Development Process*. Hershey : Idea Publishing, 2006.

—. **2000.** Process Modeling for Object Oriented Analysis using BORM Object Behavioral Analysis. *Proceedings of Fourth International Conference on Requirements Engineering ICRE 2000*. IEEE Computer Society Press , 2000.

Knott, Roger P., Merunka, Vojtěch a Polák, Jíří. 2003. The BORM methodology: a third-generation fully object-oriented methodology. *Knowledge-Based Systems*. 2003.

Larman, Craig. 2005. *Applying UML and Patterns*. Upper Saddle River : Prentice Hall, 2005. ISBN 0-13-148906-2.

Lhotka, Rockford. 2009. SOA Manifesto. *Rockford Lhotka Homepage*. [Online] 2009. [Citace: 5. 12 2011.] http://www.lhotka.net/weblog/SOAManifesto.aspx.

Li, Deyi, a další. 2010. On Foundations of Service Interoperation in Cloud Computing. [autor knihy] D.T. Lee, D.Z. Chen a S. (Eds.) Ying. *Lecture Notes In Computer Science Vol. 6213*. Heidelberg : Springer Verlag, 2010.

Li, Qin, Zhu, Huibiao a He, Jifeng. 2009. A Formal Perspective for Service Coordination Framework in Service Oriented Architecture. *Proceedings of 20TH AUSTRALIAN SOFTWARE ENGINEERING CONFERENCE*. IEEE Comp. Soc., 2009.

Margaria, Tiziana. 2011. Formal methods in the era of service-oriented design. *Proceedings of 35TH IEEE ANNUAL INTERNATIONAL COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE*. IEEE Comp. Soc., 2011.

Martin, Fowler. 2002. *Patterns of Enterprise Application Architecture*. Boston : Addison-Wesley, 2002. ISBN 978-0321127426.

McCarthy, John. 1996. REMINISCENCES ON THE HISTORY OF TIME SHARING. [Online] 9. 9 1996. [Citace: 18. 9 2010.] http://www-formal.stanford.edu/jmc/history/timesharing/timesharing.html.

Merunka, V., Brožek, J. a Nouza, O. 2008. Automated Model Transformations Using the C.C Language. *Lecture Notes in Business Information Processing*. 2008, Sv. 10.

Merunka, Vojtěch. 2008. *Objektové modelování*. Praha : Alfa, 2008. ISBN 978-80-87197-04-2.

Merunka, Vojtěch, a další. 2007. Object-Level Modeling and Testing Education - Daskalos Experience. *Proceedings of the 3rd South East European Workshop on Formal Methods*. 2007.

Merunka, Vojtěch, Brožek, Jiří a Merunková, Iveta. 2010. Organization Modeling and Simulation Using BORM Approach. *Lecture Notes in Business Information Processing*. 2010, Sv. 63.

Methodologies Corporation. 2011. Discovery and Analysis Model Language Specifications. [Online] 2011. [Citace: 10. 1 2012.] http://www.sparxsystems.com/downloads/whitepapers/SOMF-2.1-Discovery-and-Analysis-Model-Language-Specifications.pdf.

—. **2011.** Logical Design Model Language Specifications. [Online] 2011. [Citace: 10. 1 2011.] http://www.sparxsystems.com/downloads/whitepapers/SOMF-2.1-Logical-Design-Model-Language-Specifications.pdf.

Mylopoulos, John. 2003. Information Systems Analysis and Design. [Online] 2003. [Citace: 2010. 7 18.] http://www.cs.toronto.edu/~jm/340S/PDF2/Styles2.pdf.

Object Management Group. 2008. Common Object Request Broker Architecture (CORBA/IIOP) Specification. [Online] 2008. [Citace: 15. 8 2010.] http://www.omg.org/spec/CORBA/3.1/.

—. **2010.** Introduction To OMG's Specifications. [Online] 2010. [Citace: 14. 9 2010.] http://www.omg.org/gettingstarted/specintro.htm.

OMG BPMN. 2010. BPMN 2.0 Beta 2. *OMG Specifications*. [Online] 2010. [Citace: 15. 9 2010.] http://www.omg.org/spec/BPMN/2.0/Beta2/.

OMG MDA. 2010. MDA Specifications. [Online] 2010. [Citace: 5. 10 2010.] http://www.omg.org/mda/specs.htm.

Oracle. 2010. Java Remote Method Invocation. [Online] 2010. [Citace: 22. 8 2010.] http://download.oracle.com/javase/6/docs/technotes/guides/rmi/index.html.

Pecinovský, Rudolf. 2007. *Návrhové vzory.* Brno : Computer Press, 2007. ISBN 9788025115824.

Peterka, Jiří. 2005. Vývoj výpočetního modelu. *Přednáška z předmětu Počítačové sítě.* [Online] 2005. [Citace: 12. 7 2010.] http://www.earchiv.cz/l212/gifs/site30_21.pdf.

Reenskaug, Trygve. 1979. Models-Views-Controllers. [Online] 1979. [Citace: 5. 10 2010.] http://heim.ifi.uio.no/~trygver/1979/mvc-2/1979-12-MVC.pdf.

Rubin, Kenneth S. a Goldberg, Adele. 1992. Object Behavior Analysis. *Communications of the ACM - Special issue on analysis and modeling in software development.* 1992, Sv. 35, 9.

Shalloway, Alan a Trott, James R. 2001. *Design Patterns Explained, Second Edition. A New Perspective on Object-Oriented Design.* Reading : Addison-Wesley, 2001. ISBN 0-321-24714-0.

SOA Manifesto. 2009. [Online] 2009. [Citace: 25. 9 2010.] http://www.soa-manifesto.org/.

Sparx. 2011. SOMF. *Sparx.* [Online] 2011. [Citace: 5. 12 2011.] http://www.sparxsystems.com/somf.

Štumpf, Jindřich. 2006. Proč SOA nemá alternativu. *IT/SYSTEM 10/2006.* [Online] 2006. [Citace: 15. 9 2010.] http://www.systemonline.cz/sprava-it/proc-soa-nema-alternativu.htm.

van der Aalst, Wil M. P. 2010. Business Process Simulation Revisited. *Lecture Notes in Business Information Processing.* 2010, Sv. 63.

van der Aalst, Wil M. P., a další. 2009. Service Interaction: Patterns, Formalization, and Analysis. *Lecture Notes in Computer Science*. 2009, Sv. 5569.

van der Aalst, Wil M. P., ter Hofstede, Arthur H. M. a Weske, Mathias. 2003. Business Process Management: A Survey. *Lecture Notes in Computer Science*. 2003, Sv. 2678.

Seznam vyobrazení a tabulek

Obrázky

Tabulky