

# **Obsah**

<b>Predhovor.....</b>	<b>5</b>
<b>Úvod.....</b>	<b>7</b>
<b>1 Funkčné versus procesné riadenie podniku.....</b>	<b>9</b>
1.1 Organizačné útvary a ich funkcie .....	9
1.2 Podnikové procesy .....	22
1.2.1 Vývojové diagramy .....	25
1.2.2 BPMN.....	36
1.2.3 Procesné mapy.....	48
1.3 Porovnanie funkčného a procesného prístupu k riadeniu podniku .....	50
<b>2 Servisne orientovaná architektúra a jej základné charakteristiky .....</b>	<b>59</b>
2.1 Východiskové pojmy a fakty.....	59
2.1.1 Servisne orientovaná architektúra .....	59
2.1.2 Údaje, informácie, znalosti.....	59
2.1.3 Informačný systém a jeho architektúra .....	62
2.2 SOA – základné princípy a charakteristiky .....	68
<b>3 Základné charakteristiky webových služieb.....</b>	<b>81</b>
3.1 Webové služby – základné fakty.....	81
3.2 Webové služby a používateľské rozhranie .....	86
3.3 Klasifikácia webových služieb podľa potrieb SOA.....	88
3.4 Príklad použitia modelov služieb pri automatizácii podnikového procesu .....	95
<b>4 Životný cyklus informačného systému na báze SOA.....</b>	<b>111</b>
<b>5 Servisne orientovaná analýza.....</b>	<b>115</b>
<b>6 Najdôležitejšie otvorené štandardy v SOA .....</b>	<b>121</b>
6.1 XML .....	121
6.2 XSD .....	128
6.3 WSDL .....	152

<i>6.4 UDDI – registre služieb.....</i>	163
<i>6.5 SOAP.....</i>	166

## **Predhovor**

Procesné riadenie podniku predstavuje moderný prístup k riadeniu podniku, ktorý preukázaťne umožňuje dosahovať vyššiu efektivitu vnútropodnikových postupov a tokov pracovných činností, čo sa v pozitívnom zmysle prejavuje aj na konečnom výstupe podniku určenom pre konečného spotrebiteľa alebo zákazníka, ako aj na celkovej flexibilite podniku ako základnej podmienke prežitia v zložitom trhovom prostredí. Servisne orientovaná architektúra je spôsob budovania informačných systémov, ktorý umožňuje „vyťažiť“ čo najviac z procesne orientovaného manažérskeho prístupu a podporuje tvorbu systémov, ktoré prínosy tohto prístupu ešte viac umocňujú.

Publikácia zrozumiteľným spôsobom predkladá pomerne zložité problematiku servisne orientovanej architektúry informačných systémov v nadväznosti na teóriu o podnikových procesoch. Jej zámerom je, aby plnohodnotne poslúžila aj laikom, resp. ľuďom s iba základnými znalosťami zo sveta IT, a nielen odborníkom na softvérové inžinierstvo a architektúru informačných systémov, a to ako vhodný študijný materiál, ktorý im nenáročnou a „ľahko stráviteľnou“ cestou priblíži podstatu a význam spomenutej problematiky. Publikácia okrem toho zavádzajúca a definuje viacero nových pojmov, čo je možné považovať za jej príspevok k terminológii príslušného vedného odboru. V kapitole pojednávajúcej o funkčnom a procesnom riadení podniku autor vynaložil veľké úsilie na to, aby osvetlil základné chyby pri zostavovaní jednotlivých typov podnikových diagramov, a tak táto kapitola predstavuje praktický a spoľahlivý návod, ktorého sa dá pridržiavať pri ich tvorbe. V kapitole o servisne orientovanej analýze navrhujeme a prezentujeme logickú postupnosť krokov, ktoré by táto etapa životného cyklu servisne orientovaného systému mala obsahovať, aby v nej boli zachované princípy procesného riadenia podniku a poznatky z teórie podnikových procesov. Postup pre realizáciu servisne orientovanej analýzy je rozpracovaný nielen teoreticky, ale je detailne ilustrovaný na konkrétnom príklade a ide o osobitý prínos autora pre problematiku servisne orientovanej architektúry.

Na záver tohto krátkeho predhovoru by autor rád vyjadril presvedčenie, že táto publikácia naplní túžbu jej čitateľa po poznaní, obohatí jeho znalosti zo sveta IT o nové a zaujímavé skutočnosti a aj zložité poznatky mu podá nenáročnou a ľahko prijateľnou cestou.

*Autor*

## Úvod

Publikácia je rozčlenená do šiestich hlavných kapitol, ktoré sa podľa potreby ďalej členia na podkapitoly. Prvá kapitola sa venuje podstate dvoch základných prístupov k riadeniu podniku – funkčnému a procesnému prístupu. Poslaním podnikového informačného systému je podporovať prácu manažmentu podniku, ale aj ostatných zamestnancov, pričom musí zohľadňovať vzájomné väzby medzi nimi, ich kompetencie a povinnosti. Návrh informačného systému preto musí vždy vychádzať z detailnej analýzy konkrétneho podniku, jeho štruktúry a spôsobu riadenia. V prvej kapitole sa preto detailne oboznámime s rozličnými *zobrazovacími technikami*, ktoré slúžia na prehľadné znázornenie štruktúry podniku, vymenovanie funkcií jeho zamestnancov a znázornenie vzájomných vzťahov medzi týmito funkciami, ako aj tokov a nadväzností rutinne vykonávaných pracovných činností. Medzi zobrazovacie techniky môžeme zaradiť najmä *diagramové techniky* (sem patrí organizačný diagram, hierarchický diagram funkcií, vývojový diagram, Business Process Model and Notation diagram a procesná mapa), ale aj niekoľko *tabuľkových techník* (do tejto kategórie patrí relačná matica, RACI matica a rozhodovacia tabuľka). Pri diagramových technikách je výsledkom modelovania diagram, t. j. schéma zostavené z vopred definovaných symbolov pri dodržaní zodpovedajúcich pravidiel tvorby toho-ktorého diagramu. Pri tabuľkových technikách je výsledkom modelovania tabuľka, t. j. dvojrozmerná štruktúra pozostávajúca z riadkov a stĺpcov s presným významom.

V druhej kapitole sa oboznámime s podstatou servisne orientovanej architektúry ako jedného z prístupov k budovaniu podnikových informačných systémov, ale aj k ich vzájomnému prepájaniu. Ide totiž o veľmi efektívny spôsob, ako vzájomne prepájať rozličné aplikácie, medzi ktorými môžu byť staršie aj novšie, ako aj postavené na odlišných technológiach. Pred každú z takýchto aplikácií sa postaví webová služba, ktorá je schopná prijímať požiadavky od iných aplikácií a transformovať ich do podoby, v akej ich príslušná aplikácia očakáva. Webová služba teda pre aplikáciu sprostredkúva komunikáciu s okolitým svetom (plní funkciu „tlmočníka“). SOA (t. j. servisne orientovaná architektúra) môže byť veľmi užitočná tiež pri databázach, ktoré musia byť v prevádzke dlhé obdobie (desiatky rokov). Môže ísť o evidenciu zamestnancov, evidenciu občanov alebo rozličných dokumentov a údajov, ktoré treba dlhodobo archivovať a prístup k nim potrebujú viaceré systémy. Prcd aplikáciu, ktorá by do takejto databázy inak nebola schopná vstúpiť, sa postaví webová služba, ktorá jej sprostredkuje prístup, a vďaka tomu sme schopní pomerne rýchlo a efektívne prepájať aplikácie na dátovej úrovni. V tejto monografii sa ale namiesto systémovej integrácie budeme

zaoberať skôr budovaním informačného systému od základu na princípoch servisne orientovanej architektúry so zameraním na fázy, ktoré predchádzajú programovaniu (implementáciu).

V tretej kapitole sa zameriame na problematiku webových služieb, ktoré sú základnými stavebnými jednotkami systému postaveného na princípoch SOA. Informačný systém podľa konceptu SOA je množina služieb schopných vzájomnej spolupráce. Uvedieme si klasifikáciu týchto služieb podľa potrieb SOA a ich použitie ukážeme na konkrétnom príklade zameranom na automatizáciu podnikového procesu s názvom „*Objednávanie materiálu*“.

V štvrtej kapitole sa zameriame na životný cyklus systémov vybudovaných podľa princípov SOA a budeme stručne charakterizovať jednotlivé fázy tohto cyklu. V piatej kapitole sa budeme detailnejšie venovať prvotnej fáze, ktorou je servisne orientovaná analýza, a uvedieme postupnosť krokov, ktoré by túto fázu mali tvoriť, ak mienime budovať systém v súlade s potrebami procesného riadenia podniku.

Šiesta kapitola je rozsahom najväčšia a v jednotlivých podkapitolách sa venuje najdôležitejším otvoreným štandardom používaným v SOA. Týmito štandardmi sú:

- jazyk XML (Extensible Markup Language), ktorý slúži na štruktúrovaný zápis údajov;
- jazyk XSD (XML Schema Definition), ktorý slúži na tvorbu šablón predpisujúcich presnú štruktúru pre XML dokumenty;
- jazyk WSDL (Web Services Description Language), ktorý slúži na strojovo čitateľný opis spôsobu, ako sa s touto službou dá spojiť;
- štandard UDDI (Universal Description, Discovery and Integration), ktorý slúži na tvorbu centrálnych registrov služieb, vďaka ktorým sa služby môžu vzájomne vyhľadávať;
- protokol SOAP (Simple Object Access Protocol), ktorý slúži na zápis správ prenášaných medzi službami, vďaka čomu služby komunikujú.

## Kapitola 1

### Funkčné versus procesné riadenie podniku

V tejto kapitole sa oboznámime s dvoma základnými prístupmi k riadeniu podnikov, a to s historicky starším, *funkčným*, a novším, *procesným* riadením. Najskôr však musíme zadefinovať niekoľko základných pojmov, ktoré s touto problematikou súvisia a predstavujú východisko k tomu, aby sme jej správne porozumeli.

#### 1.1 Organizačné útvary a ich funkcie

Pod pojmom **organizácia** budeme chápať, ako uvádza Schmidt, „*subjekt – právnickú osobu, ktorá vykonáva určitý druh činnosti s cieľom dosiahnuť finančný, hospodársky, spoločenský alebo všeobecne prospěšný efekt*“ [SCH10]. Ako **podnik** potom môžeme označiť organizáciu, ktorej primárnym cieľom je tvorba zisku.

Podniky môžeme klasifikovať podľa viacerých hľadísk:

- **podľa vlastníctva:**

- štátne,
- súkromné,
- zmiešané – ide o kombináciu štátneho a súkromného vlastníctva,

- **podľa veľkosti:**

- mikropodniky (od 1 do 9 zamestnancov),
- malé podniky (od 10 do 49 zamestnancov),
- stredné podniky (od 50 do 249 zamestnancov),
- veľké podniky (od 250 zamestnancov),

- **podľa predmetu činností:**

- *výrobné podniky* – do tejto kategórie patria priemyselné podniky, stavebné podniky, polnohospodárske podniky, lesnícke podniky a pod.,
- *podniky služieb* – ide o dopravné podniky, obchodné podniky, kultúrno-vzdelávacie podniky, finančné podniky, podniky cestovného ruchu a pod.

Výrobné podniky môžeme podľa stupňa výroby rozčleniť na dve skupiny, a to *podniky prvovýroby* a *podniky druhovýroby*.

- *podniky pravovýroby* – ich charakteristickým znakom je, že získavajú statky z prírody a následne ich predávajú (napr. rybolov, ťažba dreva, zber divorastúcich plodín, pestovanie poľnohospodárskych plodín, chov hospodárskych zvierat a pod.),
- *podniky druhovýroby* – nakupujú a následne spracúvajú to, čo vytvorili podniky pravovýroby, a vytvárajú z toho pokročilejšie typy výrobkov (napr. textil, nábytok, potraviny a pod.).

Každý podnik má svoju **organizačnú štruktúru**, ktorá vyjadruje, z akých organizačných útvarov (napr. oddelenia, úseky, divízie a pod.), resp. z akých pracovných pozícii (napr. kontrolór výroby, skladník, mzdový účtovník, všeobecný účtovník a pod.) podnik zostáva a aké sú vzájomné hierarchické vzťahy medzi nimi. Tieto vzťahy môžu vyjadrovať vzájomnú rovnocennosť organizačných útvarov, ale aj ich vzájomnú podriadenosť či nadriadenosť.

S organizačnou štruktúrou úzko súvisia tri základné aspekty každého podnikateľského subjektu, ktoré uvádzajú Stašák vo svojej publikácii *Modelovanie systému riadenia ekonomických objektov*. Medzi tieto aspekty patria [STA10]:

- a. **základná orientácia** – vyjadruje, čo je základným cieľom podnikania danej firmy (napr. môže ísiť o výrobný podnik, ktorý sa zaobrává výrobou úžitkového skla a jeho cieľom je zabezpečiť si dominantné postavenie na trhu s úžitkovým sklom);
- b. **predmetné oblasti podnikania** – ide o definíciu oblastí, prostredníctvom ktorých chce daná firma naplniť cicl svojej predmetnej orientácie (napr. výroba olovnatého skla);
- c. **činnosti**, resp. **aktivity**, ktoré je potrebné vykonávať, aby sa napĺňali ciele predmetných oblastí a základnej orientácie podnikania firmy. Pri funkčnom riadení podniku sú tieto činnosti reprezentované funkciami jednotlivých organizačných útvarov, zatiaľ čo pri procesnom riadení ide najmä o podnikové procesy v napojení na tieto funkcie.

Existujú rozličné typy organizačných štruktúr, medzi ktoré patria najmä:

1. *Jednoduchá organizačná štruktúra malého podniku*
2. *Líniová organizačná štruktúra*
3. *Línovo-štábna organizačná štruktúra*
4. *Divisionálna organizačná štruktúra*
5. *Maticová organizačná štruktúra*

Viac o týchto štruktúrach sa dá dočítať v [JUR18]. Spomedzi týchto organizačných štruktúr je najčastejšie sa vyskytujúcou a najreprezentatívnejšou **líniová štruktúra**, resp. **líniovo-štábna**, ktorá vzniká rozšírením línievej o štábne útvary. Ide o poradné útvary, ktorých úlohou je poskytovať vedúcemu podklady pre rozhodovanie, vypracovať pre neho rozličné štatistiky a analýzy, na základe ktorých sa môže rozhodovať, no za jednotlivé rozhodnutia zodpovedá on sám. Vedúci sa teda môže, ale nemusí riadiť odporúčaniami štábneho útvaru. Ďalej platí, že štábny útvar pod sebou nikdy nemá podriadené útvary a vždy prislúcha iba k jednému vedúcemu, pre ktorého predstavuje pomoc a odbremenenie pri jeho rozhodovaní.

Základnou myšlienkou línievej organizačnej štruktúry je, že každý podriadený (pracovník alebo organizačný útvar) má iba jedného nadriadeného (pracovníka alebo organizačný útvar). Výhodou tejto štruktúry je najmä presné vymedzenie vzťahov. Vždy je jasné, kto je nadriadený a kto je podriadený. Nevýhodou je, že pri rýdzo línievej štruktúre je značne zaťažený vedúci, ktorý sa pri rozhodovaní nemôže oprieť o štábny útvar a celá ťarcha rozhodovania spočíva iba na ňom. Preto musí mať dôkladný prehľad o všetkých jemu podriadených aktivitách. Tento problém rieši už spomínaná líniovo-štábna štruktúra, umožňujúca do určitej miery odbremeniť vedúcich, ktorí to potrebujú.

Vhodným nástrojom na znázornenie organizačnej štruktúry podniku je **organizačný diagram**. Ako napovedá už jeho názov, takýto diagram sa nemusí použiť len v súvislosti s podnikom, ale pri akomkoľvek type organizácie (napr. škola, mestský úrad, výskumný ústav a pod.). Jednotlivé organizačné jednotky (oddelenia, úseky, sekcie či konkrétnie pomenované pracovné pozície – napr. skladník, grafický návrhár a pod.) sú v tomto diagrame reprezentované vhodne zoradenými obdlžníkmi, ktoré sú poprepájané neorientovanými spojnicami (t. j. čiarami bez vyznačeného smeru). Každý obdlžník musí obsahovať názov príslušného organizačného útvaru a jeho číslo. Toto číslo vyjadruje presnú pozíciu útvaru v hierarchii útvarov. Generálny riaditeľ podniku zvykne mať číslo 0, ktoré vyjadruje, že ide o nultú, a teda najvyššiu, úroveň riadenia. Pod ním sa zvyknú nachádzať najdôležitejšie oddelenia, ako sú napr. oddelenie nákupu, oddelenie výroby, oddelenie predaja a pod. Tieto oddelenia sa zvyknú číslovať postupne od jednotky, t. j. 1, 2, 3 atď., pričom predstavujú prvú hierarchickú úroveň riadenia, ktorá je nižšia ako nultá úroveň reprezentovaná riaditeľom. Tieto oddelenia sa podľa potreby môžu členiť na podútvary (napr. úseky). Ak napr. oddelenie 1 pozostáva z dvoch úsekov, je vhodné tieto úseky očíslovať ako 1.1 a 1.2. Z očislovania je zrejmé, že úseky 1.1 a 1.2 sú navzájom rovnocenné (teda medzi nimi nie sú vzťahy nadradenosť a podradenosť) a obidva sú

podriadené oddeleniu s číslom 1. Keďže ide o dve čísla oddelené bodkou, je zrejmé, že ide o druhú hierarchickú úroveň riadenia.

Dôležité je tiež dodržať jednotnosť terminológie na jednotlivých hierarchických úrovniach. Napr. ak je prvá hierarchická úroveň tvorená oddeleniami, druhá úroveň je tvorená úsekmi a tretia sekciami, potom je všetko v poriadku, pretože pomenovávanie jednotlivých organizačných útvarov je systematické a z názvu každého útvaru je ihned zrejmá jeho príslušnosť ku konkrétnej hierarchickej úrovni riadenia.

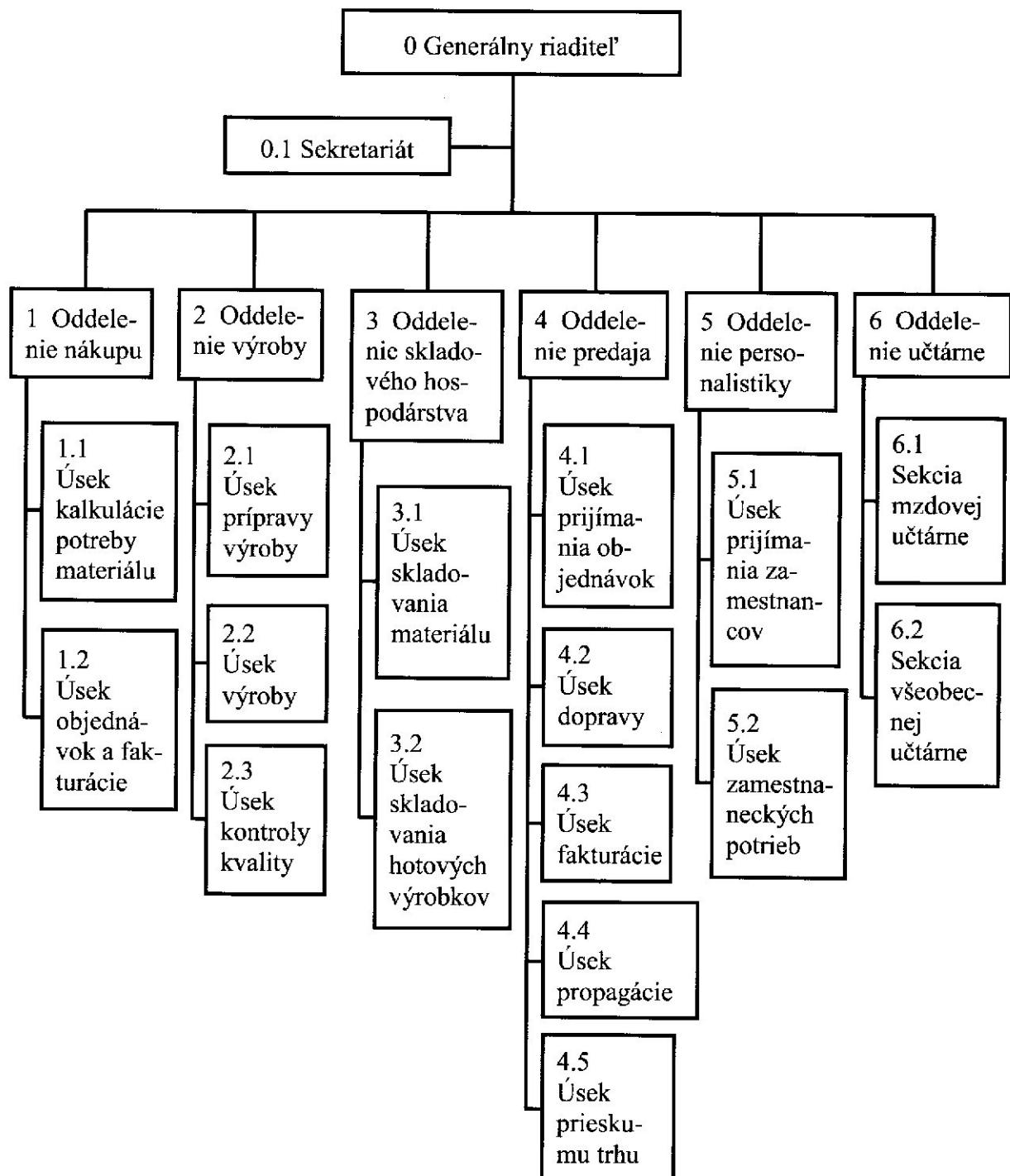
Príklad organizačného diagramu výrobného podniku s dodržaním všetkých zásad je uvedený na obr. 1. Prvá hierarchická úroveň riadenia v tomto podniku je tvorená šiestimi oddeleniami, ktoré sa podľa potreby členia na im podriadené útvary – úseky.

Každý prvok organizačnej štruktúry podniku, ktorý je znázornený na organizačnom diagramе, musí v tomto podniku plniť nejakú špecifickú funkciu, resp. funkcie. Pod **funkciou organizačného útvaru** pritom rozumieme náplň činnosti, zmysel či poslanie v organizácii. Funkcia teda predstavuje určitú činnosť, ktorá je vykonávaná konkrétnym útvarom, pričom nejde o jednorazovú činnosť, ale o takú, ktorá je vykonávaná na rutinnej báze. V tejto súvislosti je potrebné zdôrazniť, že v názve každej funkcie musí byť jasne a jednoznačne formulovaná činnosť, ktorú táto funkcia reprezentuje. Ak by sme napr. funkciu dali názov „*materiál*“, potom by nebolo jasné, aká činnosť sa má s týmto materiálom vykonat. Máme na mysli „*uskladňovanie materiálu*“? Alebo „*vyskladňovanie materiálu*“? Alebo „*zaevídovanie aktuálneho množstva materiálu na sklade do databázy*“ či „*kontrolu kvality materiálu*“? Je evidentné, že existuje mnoho činností, ktoré sa dajú s materiálom vykonávať, a preto by takýto názov funkcie nevyjadroval jednoznačne, o akú činnosť v prípade konkrétneho organizačného útvaru ide. To isté by platilo v prípade, ak by sme funkciu nazvali napr. iba „*objednávka*“, „*faktúra*“ alebo „*sklad*“. Takéto názvy funkcií sú teda neprípustné.

Funkcie zabezpečované jednotlivými organizačnými útvarmi v podniku sa zapisujú do tzv. **hierarchického diagramu funkcií**. Tento diagram teda predstavuje zoznam všetkých funkcií vykonávaných v podniku, pričom tieto funkcie sú v ňom zaznamenané hierarchickým spôsobom. To znamená, že tak ako existuje určitá hierarchia medzi organizačnými útvarmi, rovnako existuje aj hierarchia medzi jednotlivými funkciami, ktoré môžu byť vzájomne rovnocenné alebo môžu byť nadradené, resp. podradené.

Organizačný diagram je svojou povahou taktiež hierarchický diagram, pretože zobrazuje vzťahy rovnocennosti alebo podriadenosti, resp. nadriadenosti medzi jeho prvkami,

ktorými sú jednotlivé organizačné útvary. Mohli by sme ho teda oprávnenie nazývať aj hierarchický organizačný diagram. Slovo „hierarchický“ sa ale na rozdiel od hierarchického diagramu funkcií v jeho názve nezaužívalo, a tak sa zvykne nazývať len organizačný diagram.

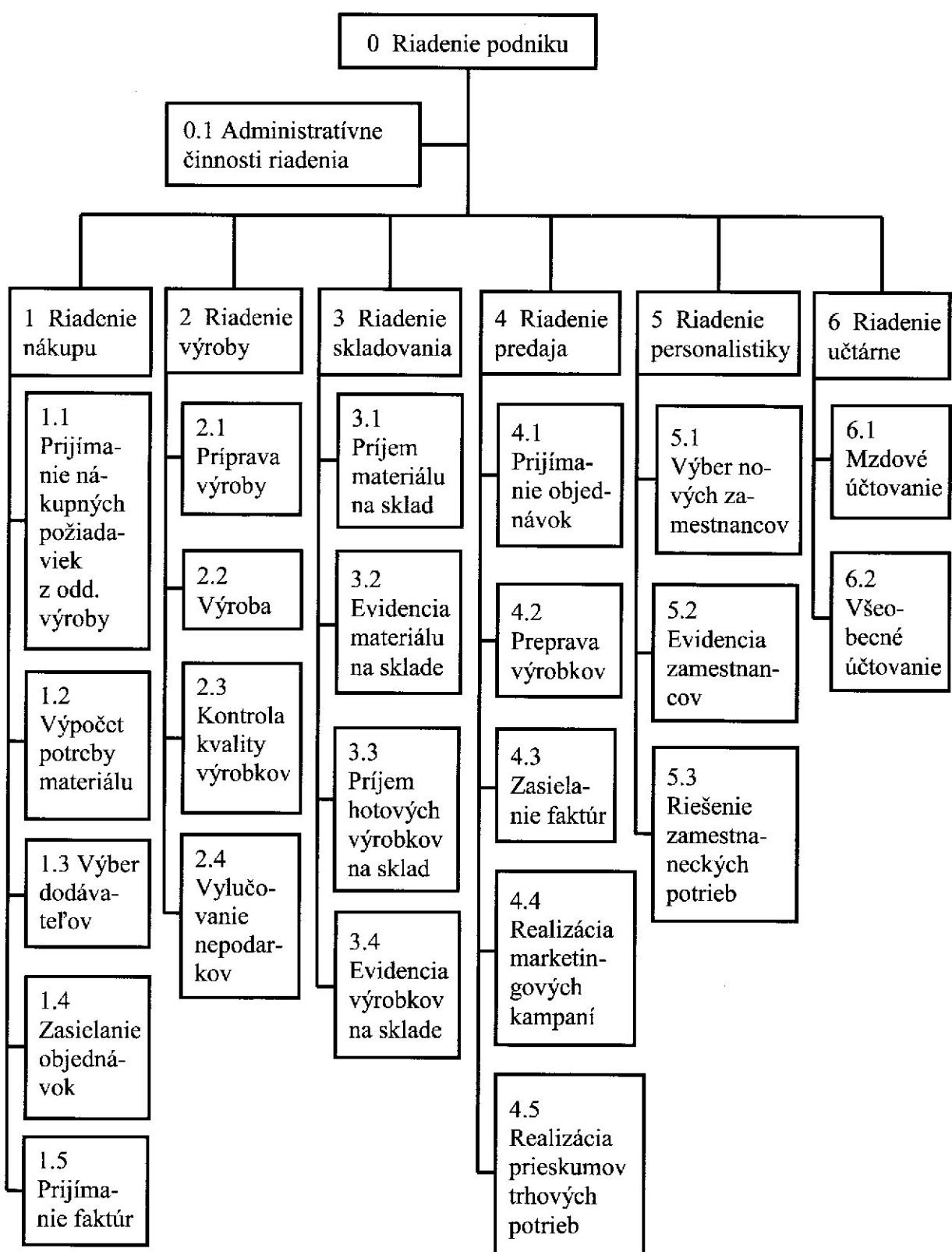


Obr. 1: Organizačný diagram [Zdroj: autor]

Vráťme sa však k hierarchickému diagramu funkcií. Jednotlivé funkcie sa v tomto diagrame zakresľujú ako obdĺžniky, ktoré sú popredávané neorientovanými spojnicami. Každý obdĺžnik musí obsahovať presný názov funkcie a jej číslo. Pri číslovaní funkcií v hierarchickom diagrame funkcií platia rovnaké zásady ako pri číslovaní organizačných útvarov v organizačnom diagrame. Nultá funkcia zvyčajne zodpovedá riaditeľovi podniku a zvykne sa formulovať ako „*riadenie podniku*“ alebo iba „*riadenie*“. Vo všeobecnosti platí zásada, že ak je určitá funkcia nadadená iným funkciám, potom sa zvykne formulovať ako riadenie „niečoho“, čo súhrnnne pomenováva priamo podriadené činnosti alebo ich určitým spôsobom kategorizuje. Napr. ak máme funkcie 2.1 *Výber optimálnych dodávateľov*, 2.2 *Zasielanie objednávok*, 2.3 *Evidencia zaslaných objednávok do databázy* a 2.4 *Prijímanie nákupných požiadaviek z oddelenia výroby*, potom môžeme funkciu, ktorá je im nadadená, pomenovať napr. 2 *Riadenie nákupu*, pretože slovo „nákup“ súhrnnne kategorizuje všetky tieto činnosti.

Príklad hierarchického diagramu funkcií, ktorý prináleží k organizačnému diagramu na obr. 1, je uvedený na obr. 2. Na tomto príklade si môžeme všimnúť, že počet obdĺžnikov (t. j. funkcií) v hierarchickom diagrame funkcií sa nerovná počtu obdĺžnikov (t. j. organizačných útvarov) v organizačnom diagrame. Je to tak preto, lebo každý organizačný útvar musí mať minimálne jednu funkciu, no môže ich mať aj viacero. Hierarchický diagram funkcií preto spravidla býva rozsiahlejší z hľadiska počtu prvkov ako organizačný diagram. Ďalej si všimnime, že niektoré funkcie v tomto ilustratívnom diagrame majú pomerne dlhé viacslovne názvy. Je dobrú praxou každú činnosť (t. j. funkciu) jednoznačne a jasne špecifikovať, aby nedochádzalo k nepochopeniu jej náplne a k nechceným obsahovým duplicitám alebo prekryvom medzi funkciami v diagrame.

Pri pohľade na hierarchický diagram funkcií nie je vidno, ktorá funkcia prináleží ktorému organizačnému útvaru z organizačného diagramu. Hierarchický diagram funkcií zobrazuje iba hierarchické vzťahy medzi funkciami. Na priradenie jednotlivých funkcií z hierarchického diagramu funkcií konkrétnym organizačným útvarom z organizačného diagramu slúži špeciálna tabuľka – *relačná matica*. Táto tabuľka teda zabezpečuje prepojenie týchto dvoch diagramov a ich vzájomnú nadväznosť – hierarchický diagram funkcií má sám o sebe (t. j. bez prepojenia na organizačný diagram) iba obmedzené možnosti použitia, pretože vďaka nemu sice vieme, aké činnosti sa v tom-ktorom podniku vykonávajú, ale nehovorí nič o tom, kto ich má na starosti. Z praktického hľadiska možno hierarchický diagram funkcií považovať za hierarchicky usporiadaný zoznam všetkých činností vykonávaných v podniku.



Obr. 2: Hierarchický diagram funkcií [Zdroj: autor]

Ako sme už povedali, vzájomný vzťah medzi organizačnými útvarmi v organizačnom diagrame a funkciami v hierarchickom diagrame funkcií vyjadruje špeciálna tabuľka, ktorá sa nazýva **relačná matica**. Pri zostavovaní relačnej matice platí niekoľko zásad, ktoré sa musia dodržať:

- Do riadkov matice sa postupne zapisujú všetky funkcie z hierarchického diagramu funkcií vo vzostupnom poradí podľa ich číselného označenia v smere zhora nadol. Ak hierarchický diagram funkcií obsahuje  $m$  funkcií, potom má relačná matica  $m$  riadkov.
- Do stĺpcov matice sa postupne zapísu všetky organizačné útvary (t. j. všetky prvky organizačného diagramu) vo vzostupnom poradí podľa ich číselného označenia v smere zľava doprava. Ak organizačný diagram obsahuje  $n$  organizačných útvarov, potom má relačná matica  $n$  stĺpcov.
- Na priesecníkoch riadkov a stĺpcov relačnej matice, ktorá má rozmery  $m \times n$ , sa nachádzajú polia, ktoré môžu buď zostať prázdne, alebo môžu obsahovať jednu z hodnôt „V“, „K“ alebo „R“.
- Ak je priesecník  $i$ -teho riadku a  $j$ -teho stĺpca matice prázdny, pre  $i = 1, 2, \dots, m$  a  $j = 1, 2, \dots, n$ , znamená to, že organizačný útvar v  $j$ -tom stĺpci sa priamo nepodieľa na vykonávaní, kontrole vykonávania a ani na riadení funkcie v  $i$ -tom riadku.
- Ak je útvar v  $j$ -tom riadku zodpovedný za vykonanie funkcie v  $i$ -tom stĺpci, potom sa do priesecníka  $i$ -teho riadku a  $j$ -teho stĺpca zapisuje hodnota „V“ (ako „vykonáva“). Ak je tento útvar zodpovedný za kontrolu správneho vykonávania príslušnej funkcie, potom sa do priesecníka zapisuje hodnota „K“ (ako „kontroluje“). Ak útvar riadi vykonávanie funkcie, potom sa zapisuje hodnota „R“ (ako „riadi“).
- Rozdiel medzi hodnotami „R“ a „K“ spočíva v tom, že riadenie je tvrdšia forma ovplyvňovania realizácie určitej činnosti. Spadá sem zadávanie konkrétnych inštrukcií, ktoré sa musia dodržať, stanovovanie jasných mantincov, ktoré sa nesmú prekročiť, ako aj samotná kontrola. Tá zasa predstavuje dohľad na správne vykonávanie činnosti, jej využitie a vydelenie adekvátnych záverov. Pojem riadenie je teda širší a kontrola je jeho súčasťou.
- Relačná matica nemôže obsahovať nijaký riadok, ktorý by bol celkom prázdný. To by znamenalo, že funkciu v takomto riadku nikto nevykonáva. Ak takáto situácia nastane, existujú dva spôsoby, ako ju napraviť:

- Ak je táto funkcia nadbytočná, potom ju môžeme z relačnej matice odobrať (t. j. odstrániť príslušný riadok), ale v tom prípade ju musíme odobrať aj z hierarchického diagramu funkcií.
  - Ak táto funkcia má svoj význam, potom ju môžeme v relačnej matici ponechať, no musíme jej vykonávanie priradiť ku konkrétnemu organizačnému útvaru. Ak sa v organizačnom diagrame nenachádza vhodný útvar a ak si môžeme dovoliť vytvoriť nejaký nový útvar, potom ho vytvoríme. To znamená, že tento útvar vhodným spôsobom pomenujeme, očísľujeme a zaradíme do organizačného diagramu, a následne do relačnej matice pridáme nový stĺpec zodpovedajúci tomuto útvaru. Na priesecník príslušného riadka a novo pridaného stĺpca potom zapíšeme hodnotu „V“, čím došlo k odstráneniu prázdnego riadka.
- Relačná matica nemôže obsahovať stĺpec, ktorý by bol celkom prázdný. To by znamenalo, že organizačný útvar v príslušnom stĺpci nemá nič na starosti. Ak takáto situácia nastane, existujú dva spôsoby ako ju napraviť:
  - Ak je tento útvar skutočne nadbytočný, tak ho môžeme z relačnej matice odobrať (t. j. odstrániť príslušný stĺpec), ale v tom prípade ho musíme odobrať aj z organizačného diagramu.
  - Ak sa má tento útvar ponechať, potom mu musíme priradiť nejakú funkciu, ktorú bude mať na starosti. Ak sa v hierarchickom diagrame funkcií nenachádza funkcia, ktorá by svojím obsahom zodpovedala zameraniu tohto útvaru, potom je potrebné vymysliť novú funkciu. Túto funkciu vhodne zaradíme do hierarchického diagramu funkcií a v relačnej matici pre ňu vytvoríme nový riadok. Na priesecník novo vytvoreného riadka a stĺpca, ktorý bol doposiaľ prázdný, potom podľa potreby zapíšeme hodnotu „V“.
- V každom riadku relačnej matice sa musí objaviť hodnota „V“. Častou chybou je, že pri niektornej funkcií sa špecifikuje iba, kto má na starosti kontrolu jej vykonávania alebo jej riadenie, ale nešpecifikuje sa, kto je zodpovedný za priame vykonanie tejto funkcie. Ide teda o situácie, keď riadok nie je celkom prázdný, pretože je v ňom hodnota „K“ alebo „R“, ale chýba v ňom hodnota „V“.
- Veľmi dôležitou zásadou je, že **každý organizačný útvar musí vykonávať minimálne jednu funkciu, ale môže ich vykonávať aj viac, pričom maximálny počet**

funkcií pre jeden útvar nie je limitovaný (pracovné povinnosti však musia byť nastavené tak, aby nedochádzalo k preťažovaniu zamestnancov a porušovaniu zákonníka práce).

- Nemôže nastať situácia, pri ktorej dva rôzne útvary vykonávajú tú istú funkciu. To by znamenalo, že dochádza k duplicitám v prerozdelení kompetencií medzi organizačnými útvarmi, v dôsledku čoho podnik platí dva alebo viaceré organizačné útvary za to, že vykonávajú tú istú činnosť. Z toho vyplýva, že **každá funkcia smie byť vykonávaná iba jediným organizačným útvarom.**
- Pri vypĺňaní relačnej matice sa snažíme vyhnúť slovnému spojeniu „riadenie riadenia“, ktoré nemá zmysel. Dochádza k nemu vtedy, ak funkcia v určitom riadku obsahuje vo svojom názve slovíčko „riadenie“ a zároveň sa na priesčníku tohto riadka a niektorého zo stĺpcov nachádza hodnota „R“. Riadenie teda môžeme len vykonávať (hodnota „V“), no nemôžeme ho riadiť.

Príklad relačnej matice je uvedený v tab. 1. Ide o maticu prepájajúcu funkcie v hierarchickom diagrame funkcií na obr. 2 s organizačnými útvarmi v organizačnom diagramе на obr. 1. V skutočnosti je v tab. 1 zobrazený len „výsek“ z tejto matice, ktorá je príliš široká na to, aby sa zmestila na jednu stranu (sú z nej preto odstránené stĺpce zodpovedajúce organizačným útvaram 5.2, 5.3, 6, 6.1 a 6.2). Relačná matica sa totiž dá považovať za komplet-nú iba vtedy, ak obsahuje všetky funkcie z hierarchického diagramu funkcií, ako aj všetky organizačné útvary z organizačného diagramu. Ďalej si všimnime, že ak sú funkcie zapísané v riadkoch a organizačné útvary v stĺpcoch, potom je matica orientovaná na výšku. Je to tak preto, lebo celkový počet funkcií v hierarchickom diagrame funkcií býva spravidla vyšší než počet útvarov v organizačnom diagrame, a tak má matica viac riadkov ako stĺpcov. Ak by sme funkcie zapisovali do stĺpcov a útvary do riadkov, potom by matica bola orientovaná na šírku.

S funkciami organizačných útvarov súvisí aj **diagram funkčných závislostí**, ktorého úlohou je zachytiť vzájomné toky medzi funkciami z hierarchického diagramu funkcií. Tieto toky môžu byť *hmotné* (napr. hotové výrobky, materiál a pod.), *informačné* (napr. objednávky, faktúry, príkazy na úhradu, výpisu z účtu, rozličné typy inštrukcií, dokumentov a pod.) a *finančné* (všetky typy hotovostných alebo bezhotovostných platieb – napr. vyplácanie mzdy, úhrada faktúr, platenie odvodov a pod.). Diagram funkčných závislostí sa zvykne zostavovať *na globálnej úrovni* (ide o vzájomné toky medzi funkciami na prvej hierarchickej úrovni riadenia) a tiež *na čiastkových úrovniach*, kde ide o vzájomné toky medzi spravidla

rovnocennými funkciemi zameranými na určitú tematickú oblasť. Príklad globálnej úrovne je znázornený na obr. 3 [JUR18].

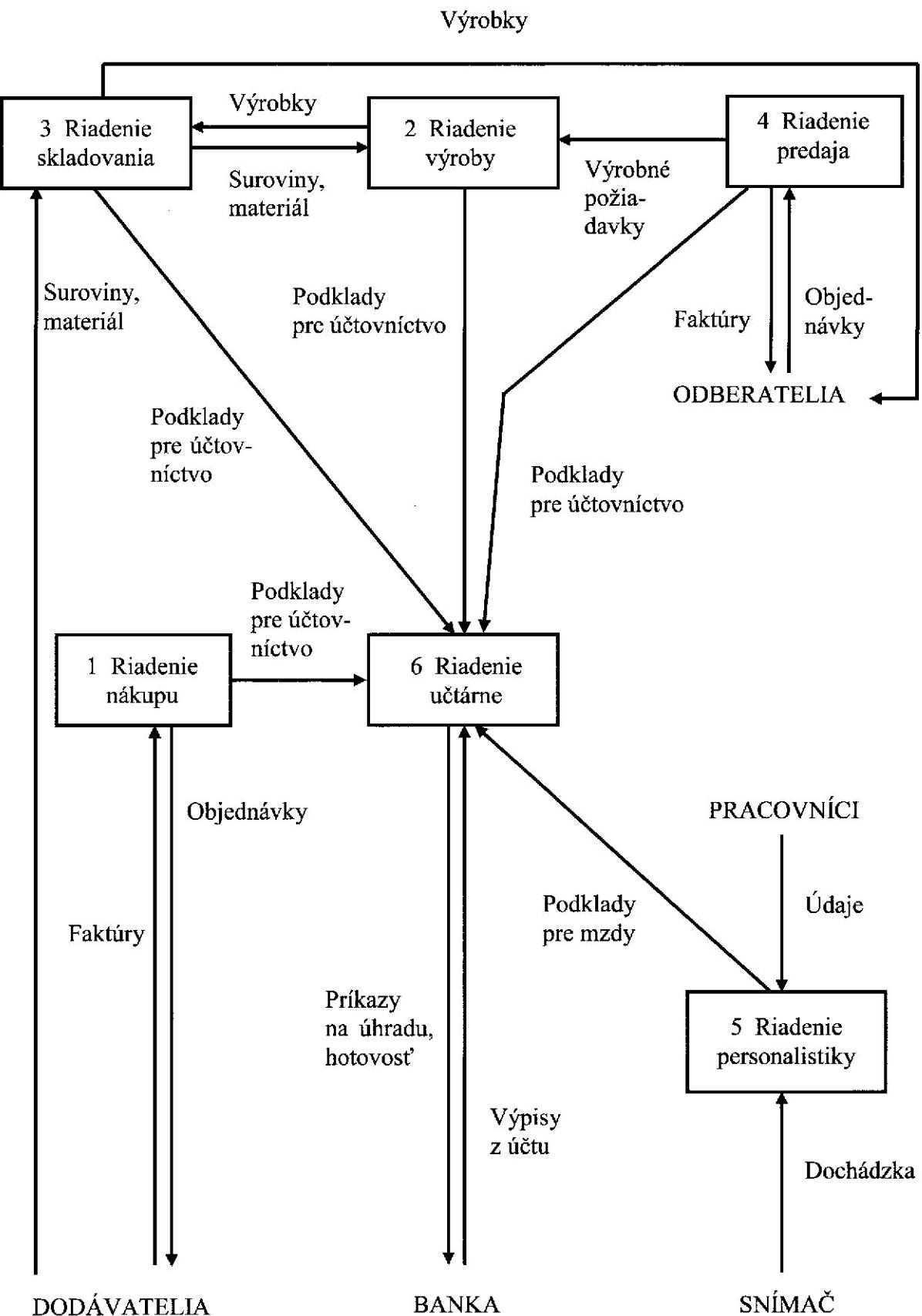
Tab. 1: **Relačná matica** [Zdroj: autor]

<b>Funkcia</b>	<b>Útvar</b>																		
0	0	0.1	1	1.1	1.2	2	2.1	2.2	2.3	3	3.1	4	4.1	4.2	4.3	4.4	4.5	5	5.1
0	V																		
0.1	R	V																	
1	K		V																
1.1			K	V															
1.2			K	V															
1.3			K		V														
1.4			K		V														
1.5			K		V														
2	K				V														
2.1					K	V													
2.2					K		V												
2.3					K			V											
2.4					K			V											
3	K							V											
3.1								K	V										
3.2								K	V										
3.3								K		V									
3.4								K		V									
4	K									V									
4.1										K	V								
4.2										K		V							
4.3										K			V						
4.4										K			V						
4.5										K				V					
5	K													V					
5.1														K	V				
5.2														K					
5.3														K					
6	K																		
6.1																			
6.2																			

Prvkami diagramu funkčných závislostí sú obdlžníky, ktoré reprezentujú funkcie organizačných útvarov, a tiež externé subjekty (napr. dodávateľ, odberateľ, banka, úrad, ministerstvá a pod.). Externé subjekty sa nezapisujú ako obdlžníky a zvyknú sa nazývať veľkými písmenami, aby boli dostatočne vizuálne odlišené od funkcií. Funkcie a externé subjekty sa zvyknú prepájať šípkami reprezentujúcimi toky, pričom každá šípka musí byť pomenovaná (jej názov pomenúva to, čo je prenášané) a musí mať znázornený smer. To znamená, že sú prípustné len orientované spojnice (šípky). Neorientované spojnice (čiary bez vyznačeného smera) sa v diagrame funkčných závislostí nesmú vyskytovať. Ak potrebujeme medzi rovnocenné funkcie zaradiť funkciu z inej oblasti alebo hierarchickej úrovne, pretože medzi nimi dochádza k určitému toku, potom sa táto funkcia znázorňuje tak, akoby išlo o externý subjekt. To znamená, že sa musí prevziať presný názov aj číselné označenie tejto funkcie podľa hierarchického diagramu funkcií, no táto funkcia sa zapisuje veľkými písmenami a nedáva sa do obdlžníka. Platí tiež zásada, podľa ktorej sa nemodelujú vzájomné toky medzi externými subjektmi.

Užitočnou pomôckou pri zostavovaní diagramu funkčných závislostí je zásada, podľa ktorej sa pri každej funkcií (t. j. pri každom obdlžníku) treba zamyslieť nad tým, či si táto funkcia vie sama zabezpečiť všetko, čo potrebuje pre svoju činnosť, a či nepotrebuje dostať nejaký vstup od niektorého zo zvyšných funkcií alebo od niektorého externého subjektu. Tiež sa treba zamyslieť nad tým, aké výstupy má funkcia poskytovať a kam sú určené (t. j. kam ich treba nasmerovať šípkou). Dodržiavanie týchto zásad je dôležité preto, aby diagram funkčných závislostí obsahoval všetky toky, ktoré má obsahovať. Nemalo by sa teda stať, že na nejaký tok zabudneme.

Globálna úroveň diagramu funkčných závislostí na obr. 3 je zostavená pre funkcie na prvej hierarchickej úrovni hierarchického diagramu funkcií na obr. 2. Všimnime si, že tieto funkcie sú prevzaté s presným názvom aj číselným označením. Šípky, ktoré znázorňujú vzájomné toky medzi jednotlivými funkciemi alebo toky medzi funkciami a externými subjektmi, je vhodné zakresľovať tak, aby sa navzájom neprekrižovali. Prekrižovanie šípok zhorešuje prehľadnosť diagramu a môže viesť k nejasnostiam. Taktiež si všimnime veľký počet informačných tokov, ktoré sú na tomto diagrame znázornené. Je to tak preto, lebo medzi jednotlivými činnosťami (t. j. funkciami) v podnikoch zvyčajne dochádza k masívnej výmene informácií. Môže ísť o rôzne požiadavky, pokyny, potvrdenia, dokumenty, výsledky údajových analýz, podkladové údaje pre analýzy a iné. Podľa relačnej matice potom vieme, ktoré organizačné útvary majú na starosti jednotlivé funkcie, a teda odkiaľ kam informácie prúdia.



Obr. 3: Diagram funkčných závislostí [Zdroj: autor]

## 1.2 Podnikové procesy

Po pojednaní o organizačných štruktúrach a funkciách organizačných útvarov môžeme pristúpiť k analýze ďalšieho dôležitého pojmu, ktorým je *podnikový proces*. Robson a Ullah definujú **podnikový proces** ako „*tok práce, postupujúci od jedného človeka k druhému, a v prípade väčších procesov pravdepodobne od jedného oddelenia k druhému*“ [ROB98]. Podľa Davenporta a Shorta: „*proces je séria logicky spojených úloh vykonávaných tak, aby prinášali definovaný výstup biznisu*“ [DAV90]. Z praktického hľadiska môžeme podnikový proces definovať ako *ustálený spôsob riešenia určitého konkrétneho opakujúceho sa problému v konkrétnom podniku*. Ide teda o ustálené a opakujúce sa činnosti prebiehajúce v konkrétnom podniku, ktoré pozostávajú z určitého počtu krokov, smerujúcich k dosiahnutiu vopred stanoveného cieľa. Ak je týmto cieľom napr. prijatie nového zamestnanca, potom môžeme hovoriť o procese prijímania nových zamestnancov, pričom takýto proces reprezentuje spôsob (postupnosť krokov), ako chce konkrétny podnik postupovať vždy, ak nastane potreba prijať nového zamestnanca. Pri realizácii krokov procesu zvyčajne dochádza k transformácii určitej množiny vstupov na množinu výstupov s tým, že tieto vstupy a výstupy môžu byť hmotné, informačné alebo finančné.

Podnikový proces je teda zovšeobecnený postup, ako chce konkrétny podnik riešiť konkrétny problém vždy, keď tento problém nastane. Tento postup by mal byť navrhnutý tak, aby počítal s rozličnými situáciami, ktoré potenciálne môžu pri riešení problému nastať, a aby predstavoval spoľahlivý návod na riešenie problému. Každá jedna realizácia procesu sa označuje ako **inštancia procesu**. Pri inštancii procesu sa postupuje presne podľa krokov procesu so zámerom dosiahnuť cieľ procesu. Predstavme si napr. proces vybavovania zákazníckej objednávky ako postupnosť krokov, pomocou ktorých chce podnik obslužiť objednávku prijatú od zákazníka. Vždy keď príde nová objednávka, začne sa nová inštancia tohto procesu. Z toho logicky vyplýva, že v jednom časovom okamihu môže byť spustených a nedokončených (t. j. rozpracovaných) aj viacero inštancií tohto istého procesu. Ako podnik postupne prijíma objednávky od svojich zákazníkov, postupne sa spúšťajú inštancie procesu vybavovania zákazníckej objednávky a zakaždým, keď sa podarí konkrétnu objednávku obslužiť, príslušná inštancia tohto procesu zaniká.

Príchod novej objednávky od zákazníka je podnetom, resp. signálom na to, aby sa spustila nová inštancia procesu, ktorý je s takýmto podnetom asociovaný. Príchod novej objednávky teda predstavuje spúšťaciu udalosť procesu. Každý proces musí mať svoju

spúšťaciu udalosť, ktorá je jasne definovaná, pretože inak by nebolo možné rozpoznať, kedy sa má proces nanovo spustiť.

V podniku sa môže realizovať množstvo procesov rozličného zamerania, pričom v jednom časovom okamihu môže byť pospúšťaných a nedokončených niekoľko inštancií každého z týchto procesov.

Každý proces musí mať svoj *jednoznačný názov* a musí mať stanovený *cieľ*. **Názov procesu** musí pomenovať činnosť, ktorú tento proces reprezentuje. **Cieľ procesu** musí byť stanovený tak, aby bol jednoznačný a merateľný. Napr. ak povieme, že chceme realizáciou procesu výroby vytvoriť výrobok, ktorý bude pekný a odolný, potom je cieľ takéhoto procesu stanovený nejednoznačne. Slovo „pekný“ je subjektívne a z pohľadu rôznych ľudí môže znamenať niečo iné. Slovo „odolný“ môže taktiež znamenať všeličo. Ak ale povieme, že chceme vytvoriť výrobok, ktorý bude mať konkrétny farebný odtieň podľa farebníka (t. j. katalógu farieb), bude schopný odolať tlaku o konkrétej sile a bude vodotesný, potom máme cieľ, ktorý je jednoznačný a jeho dosiahnutie je overiteľné.

**Aktérov podnikových procesov**, t. j. osoby, ktoré na podnikových procesoch participujú, môžeme rozčleniť do štyroch kategórií [KAI13]:

- **vlastník procesu** (process owner) – je osoba, ktorá zodpovedá za navrhnutie a zostavenie procesu tak, aby smeroval k dosahovaniu cieľov stanovených vedením podniku. Vlastník procesu zodpovedá za tvorbu, aktualizáciu a zlepšovanie dokumentov, ktoré určujú správny priebeh procesu a poskytujú návody na riešenie všetkých problematických situácií, ktoré potenciálne môžu nastaviť. Pri napĺňaní tohto poslania sa nemusí spolichátať iba sám na seba, ale môže sa opierať o podporný tím (tzv. process improvement team). Vlastník procesu je však jediný, kto má právomoc stanovovať zmeny v chode procesu a riadiť všetko, čo súvisí so zlepšovaním procesu;
- **prevádzkovateľ procesu** (process operator) – je osoba, ktorá je zodpovedná za to, aby sa naučila správny chod procesu tak, ako ho stanovil vlastník procesu. Zároveň zodpovedá za jeho chod a výsledky. Okrem zodpovednosti musí mať táto osoba aj dostatočné právomoci na to, aby mohla aktívne konať a riadiť pracovníkov pri vykonávaní jednotlivých činností, ktoré dohromady vytvárajú predmetný proces;
- **vykonávateľ procesu** (process executor) – je každá osoba, ktorá má na starosti priame vykonanie konkrétnego kroku procesu, alebo sa na jeho vykonávaní podieľa spolu s inými osobami;

- **zákazník procesu** (process customer) – je subjekt alebo osoba, ktorému, resp. ktorej sú určené výsledky procesu. Pritom môže ísť o interný subjekt (konkrétnie oddelenie alebo pracovník v tom istom podniku) alebo externý subjekt (zákazník, dodávateľ, banka, reklamná agentúra, orgány štátnej správy a pod.).

Procesy môžeme členiť podľa rozličných hľadísk:

- **Podľa dôležitosti a účelu procesu na:**

- *hlavné* – priamo sa zúčastňujú na zabezpečovaní cieľov podniku (procesy súvisiace s postupmi vo výrobe, nákupe a predaji),
- *riadiace* – nadvádzajú na hlavné procesy, zabezpečujú ich riadenie, monitorovanie a kontrolu tak, aby sa ciele podniku napĺňali v požadovanej kvalite a aby sa pritom dodržiavali všetky zákony a interné aj externé predpisy, ktoré sú pre ten-ktorý podnik zaväzné (takéto procesy reprezentujú rozličné formy kontroly dodržiavania pracovných postupov a prenosu inštrukcií, vyhodnocovania činnosti zamestnancov alebo zlepšovania efektivity a kvality),
- *podporné* – zabezpečujú aspekty, ktoré nesúvisia priamo s dosahovaním cieľov podniku, ale sú potrebné z hľadiska chodu organizácie a z hľadiska podpory pre iné procesy (procesy súvisiace s personalistikou – napr. prijímanie nových zamestnancov, zaškoľovanie zamestnancov; procesy súvisiace s účtovníctvom, bezpečnosťou či marketingom).

- **Podľa vlastníka na:**

- *interné* – ide o procesy, ktoré má manažment podniku úplne pod kontrolou, môže im prideliť vlastníka a riadiť celý ich priebeh. Takéto procesy sa v celom rozsahu uskutočňujú len v rámci jednej konkrétnej firmy,
- *externé* – ide o procesy, ktoré manažment podniku nemá úplne pod kontrolou, pretože sa na nich zúčastňujú aj externé subjekty (zákazníci, dodávatelia, dopravcovia a pod.), ktoré majú na starosti vykonanie niektorých čiastkových činností v rámci takéhoto procesu.

Proces je tvorený postupnosťou činností, ktoré v presne stanovenom poradí systematicky smerujú k tomu, aby sa vyriešil s ním asociovaný problém a naplnil sa cieľ procesu. Pritom rozlišujeme *sekvenčný tok procesu* a *paralelný tok*. **Sekvenčný tok** znamená, že

niektoré kroky procesu sú vykonávané v presne určenom poradí jeden po druhom. To znamená, že na každý nasledujúci krok takejto sekvencie (t. j. postupnosti) sa dá prejsť až po tom, keď sa dokončil bezprostredne predchádzajúci krok. **Paralelný tok procesu** znamená, že v určitom okamihu sa tok procesu rozvetví do viacerých prúdov (tzv. vlákien), ktorých vykonávanie sa spustí naraz. Každé z týchto vlákien pritom pozostáva zo sekveníc krovov s presne stanoveným poradím, pričom počet krovov v jednotlivých vláknach nemusí byť rovnaký. Keď sa dokončia činnosti vo všetkých vláknach, dochádza zvyčajne k zjednoteniu toku procesu opäť do jednej vetvy. Tým je paralelizmus ukončený. Každý proces môže mať fázy, v ktorých je jeho tok sekvenčný, a fázy, v ktorých je jeho tok paralelný. Sekvenčné toky však spravidla výrazne prevažujú. Paralelný tok v procese nie je nevyhnutnosťou, ale môže vykonávanie procesu urýchliť, a preto je vitaný.

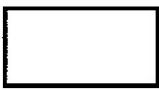
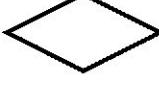
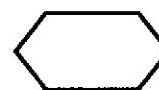
Pri každom procese musí byť jasné, v akom okamihu sa začína a kedy sa končí. Procesy sa spravidla začínajú iba v jednom bode, ale bodov, v ktorých sa končia, môže byť potenciálne viac. Jeden z týchto koncov spravidla reprezentuje tzv. **želaný koniec**. Takýto typ konca reprezentuje skutočnosť, že bol v konkrétnej inštancii procesu dosiahnutý cieľ procesu, a teda, že sa táto inštancia skončila úspešne. Okrem toho sa však v procese môže vyskytovať jeden alebo viac bodov, v ktorých môže inštancia procesu skončiť predčasne. Predčasné ukončenie inštancie znamená, že v tejto inštancii už nemôže byť dosiahnutý cieľ procesu, a teda jej pokračovanie nemá ďalej zmysel. Bod predčasného ukončenia procesu predstavuje tzv. **neželaný koniec**. Ak znázorníme postupnosť krovov, tvoriacich určitý proces, do diagramu zakresleného pomocou niektornej z vhodných diagramových techník, potom každý takýto diagram musí poviňne obsahovať jeden želaný koniec procesu a nula, jeden až  $n$  neželaných koncov.

### 1.2.1 Vývojové diagramy

Jednoduchou, ale veľmi efektívnu diagramovou technikou, slúžiacou na znázorňovanie priebehu podnikových procesov, sú **vývojové diagramy**. Táto technika je primárne určená pre programátorov na tvorbu diagramov znázorňujúcich postupnosť krovov určitého algoritmu, ktorý treba naprogramovať (napr. priebeh určitého výpočtu). Výborne sa však hodí aj na zakreslovanie podnikových procesov, ktoré sa vo svojej podstate v mnohom podobajú na algoritmy. Vývojové diagramy pozostávajú z množiny konkrétnych značiek, z ktorých každá má presne špecifikovaný význam a použitie. Množina značiek je štandardizovaná medzinárodnou normou ISO 5807:1985 *Information processing – Documentation symbols and*

*conventions for data, program and system flowcharts, program network charts and system resources charts* [ISO85]. Spomedzi rozličných značiek, ktoré sa vo vývojových diagramoch dajú použiť, sú pre účely zakresľovania priebehu podnikových procesov použiteľné najmä tieto značky (tab. 2):

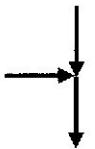
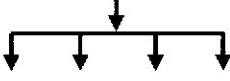
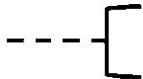
Tab. 2: Značky vývojových diagramov [Zdroj: autor]

Názov značky	Symbol	Charakteristika
Spracovanie		Ide o všeobecnú značku – v tvare obdĺžnika – na vykonanie určitej operácie (kroku procesu).
Rozhodovanie		Značkou kosoštvorca sa znázorňuje rozhodovací blok, ktorý vyjadruje rozhodovanie o ďalšom toku procesu. Na rozhodovací blok sa napájajú dve alebo viaceré vetvy, ktorými sa môže príslušný proces uberať, a ďalší postup sa stanovuje na základe splnenia či nesplnenia určitej podmienky.
Príprava		Značka šestuholníka predstavuje úpravu alebo modifikáciu určitej činnosti. Veľmi často sa používa pri cykloch s vopred známym počtom opakovaní. V tom prípade má značka dve vstupné spojnice (jednu sekvenčnú – na prvotné spustenie cyklu a druhú spätnú – na prechod na ďalšiu iteráciu) a dve výstupné spojnice (jednu smerujúcu do tela cyklu, a druhú, ktorá smeruje ku krokom nasledujúcim po ukončení cyklu).
Vstup a výstup údajov		Značka kosodlžníka slúži na označenie vstupných a výstupných operácií s údajmi – teda dodanie vstupných údajov do programu, resp. poskytnutie výstupných údajov.

Tab. 2: druhá časť

Názov značky	Symbol	Charakteristika
Ručný vstup		Značka na označenie všetkých nosičov údajov a zariadení na ručný vstup údajov (najmä svetelné pero, snímač čiarového kódu, rozličné prepínače a iné).
Interná pamäť		Značka označuje vnútornú pamäť počítača (RAM). Údaje sa z nej môžu buď získať (výstupná spojnica do značky na vstup a výstup údajov), alebo môžu byť do nej zapísané (vstupná spojnica zo značky na vstup a výstup údajov).
Začiatočná a koncová značka		Značka na vymedzenie začiatku procesu a taktiež na vymedzenie jeho konca. Písmenom „Z“ alebo „K“ v jej strede určíme, či ide o začiatočnú alebo koncovú značku.
Dokument		Značka na znázornenie zariadenia určeného na tlačený výstup (najmä tlačiareň, ale aj zariadenie na záznam snímok na mikrofilm).
Spojka		Symbol kružnice sa používa na prechod z jednej časti vývojového diagramu na inú časť (prerušenie spojnice a jej pokračovanie na inom mieste diagramu).
Spojnica		Ide o symbol určený na znázornenie toku procesu vyjadrujúci následnosť vykonávania jednotlivých jeho krokov. Spojnice je vhodné doplniť o šípky vyjadrujúce presný smer toku.
Kríženie spojníc		Spojnice, ktoré sa krížia, nemajú vzájomný vzťah. Kvôli prehľadnosti diagramu je vhodné sa kríženiu spojníc vyhnúť – na tento ...

Tab. 2: tretia časť

Názov značky	Symbol	Charakteristika
Križenie spojníc		...účel môžeme použiť párový symbol spojky, ktorým na diaľku spojíme dve miesta bez toho, aby sme medzi nimi viedli spojnicu.
Spájanie spojníc		Ide o spájanie dvoch alebo viacerých spojníc do jednej výslednej spojnice. Spojnice je nutné označiť šípkami, aby bolo jasné, ktoré spojnice sú vstupné a ktorá je výstupná.
Vetvenie spojníc		Jedna spojnice sa rozvetvuje do dvoch alebo viacerých samostatných spojníc – tok procesu sa rozdeľuje na paralelné vlákna. Spojnice musia mať smerové šípky.
Anotácia		Značka na pripájanie komentárov a opisných textov k rozličným časťam diagramu.

Pri zostavovaní vývojových diagramov sa musia dodržať najmä tieto pravidlá:

- Jeden vývojový diagram reprezentuje priebeh jedného podnikového procesu.
- Každý proces musí mať jasne znázornený začiatok a koniec. Koncov pritom môže byť viac, pričom jeden z nich predstavuje želaný koniec a ostatné predstavujú neželané konce (t. j. predčasné konce, ktoré znamenajú, že sa inštancia procesu končí neúspechom a nemá zmysel, aby ďalej pokračovala). Dve značky pre začiatok v tom istom diagrame by znamenali, že existujú dva rôzne podnety, ktoré môžu spustiť ten istý proces. To sice nie je celkom vylúčené, ale ide o pomerne zriedkavý jav.
- Pri kreslení vývojových diagramov uprednostňujeme smer zhora nadol, príp. zľava doprava.
- Spojnice sa kreslia iba vo zvislom alebo vodorovnom smere a v prípade potreby sa môžu lámať v pravom uhle (t. j. nemali by viest' šikmo) a môžu sa vetviť, spájať a ojedinele aj prekrižovať.

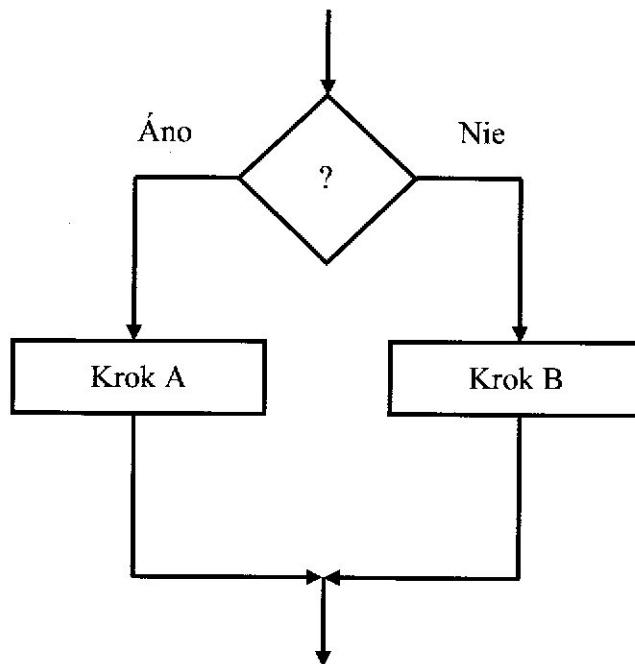
- Prekrižovanie spojníc sa neodporúča, keďže môže pôsobiť mätúco (je lepšie sa mu vyhnúť).
- Vývojové diagramy môžu obsahovať rozhodovacie bloky, ktoré reprezentujú vetvenie toku procesu podľa splnenia alebo nesplnenia určitej logickej podmienky. Z rozhodovacieho bloku vedú spojnice (zvyčajne dve), z ktorých jedna reprezentuje kladnú vetvu, ktorá vyjadruje, čo sa má robiť, ak je podmienka splnená (jej pravdivostná hodnota je *true*, t. j. *pravda*), a druhá reprezentuje zápornú vetvu, vyjadrujúcu, čo sa má robiť, ak podmienka nie je splnená (jej pravdivostná hodnota je *false*, t. j. *nepravda*). Každá vetva musí byť náležite slovne alebo číselne označená tak, aby bolo jasné, či ide o kladnú alebo zápornú vetvu.
- Každá spojnica musí mať jednoznačne znázornený smer pomocou šípky, aby nedochádzalo k pochybnostiam o jej smerovaní (t. j. musí byť orientovaná). Vo vývojových diagramoch sa teda nesmú nachádzať žiadne neorientované spojnice.
- Každá spojnica musí vždy viest' z jedného symbolu do iného, nikdy sa nesmie začínať v prázdnom priestore alebo sa v ňom končiť (t. j. nesmie viest' „odníkial“ alebo sa končiť „v ničom“).
- Pri zostavovaní vývojového diagramu treba klášť veľký dôraz na správne poradie všetkých krokov procesu. Tento diagram musí predstavovať spoločnosť a presný návod na to, ako by sa v konkrétnom podniku mala riešiť konkrétna situácia, ktorá v ňom opakovane nastáva.

Rozlišujeme viacero typov vetvení toku procesu:

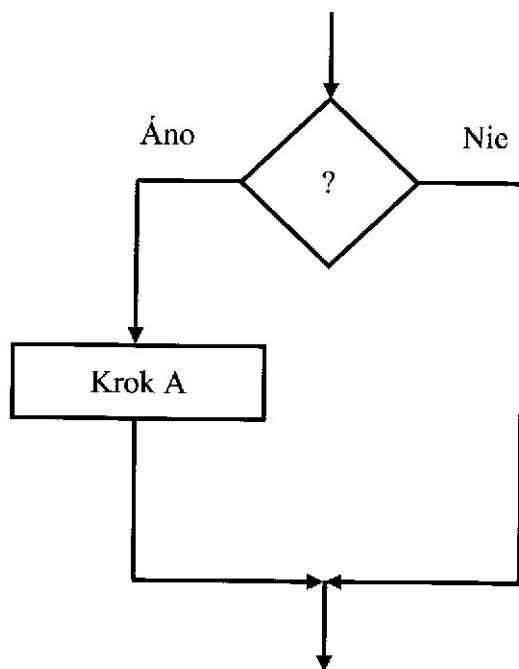
- **Jednoduché vetvenie** – tok procesu sa rozvetví na základe splnenia alebo nesplnenia jednej podmienky. Jednoduché vetvenie sa ďalej člení na:
  - *úplné* – ani jedna z vetiev nie je prázdna. To znamená, že každá z týchto vetiev obsahuje aspoň jeden krok, ktorý sa má vykonat', keď sa príslušná vetva aktivuje (počet krokov v jednotlivých vetvách pritom nemusí byť rovnaký).
  - *neúplné* – jedna z vetiev je prázdna, t. j. neobsahuje ani jeden krok. Zvyčajne ide o zápornú vetvu (teda vetvu, ktorá vyjadruje, čo sa má vykonať, ak podmienka nie je splnená). Prázdna vetva sa tak či tak musí zakresliť – nedá sa vynechať, aj keď neobsahuje žiadne kroky.

- **Viacnásobné vetvenie** – ide o zreteženie viacerých podmienok, z ktorých každá sa musí v konkrétnnej situácii samostatne vyhodnotiť, či platí alebo neplatí. Zreteženie podmienok znamená, že určitá podmienka rozvetví tok procesu na dve vetvy, pričom každá z týchto vetiev môže obsahovať ďalšiu podmienku, ktorá tok procesu ďalej rozvetvuje. Tok procesu sa teda pri viacnásobnom vetvení postupne rozdeľuje na viacero vetiev a z topologického hľadiska pripomína strom. Tieto vetvy sa podľa potrieb a špecifík konkrétneho procesu môžu neskôr znova spájať, ale nie je to nevyhnutné. Môžu nastávať situácie, pri ktorých niektorá vetva vedie až ku koncu inštancie procesu (t. j. je zakončená koncovou značkou), a preto ju nie je potrebné spájať s inými vetvami.

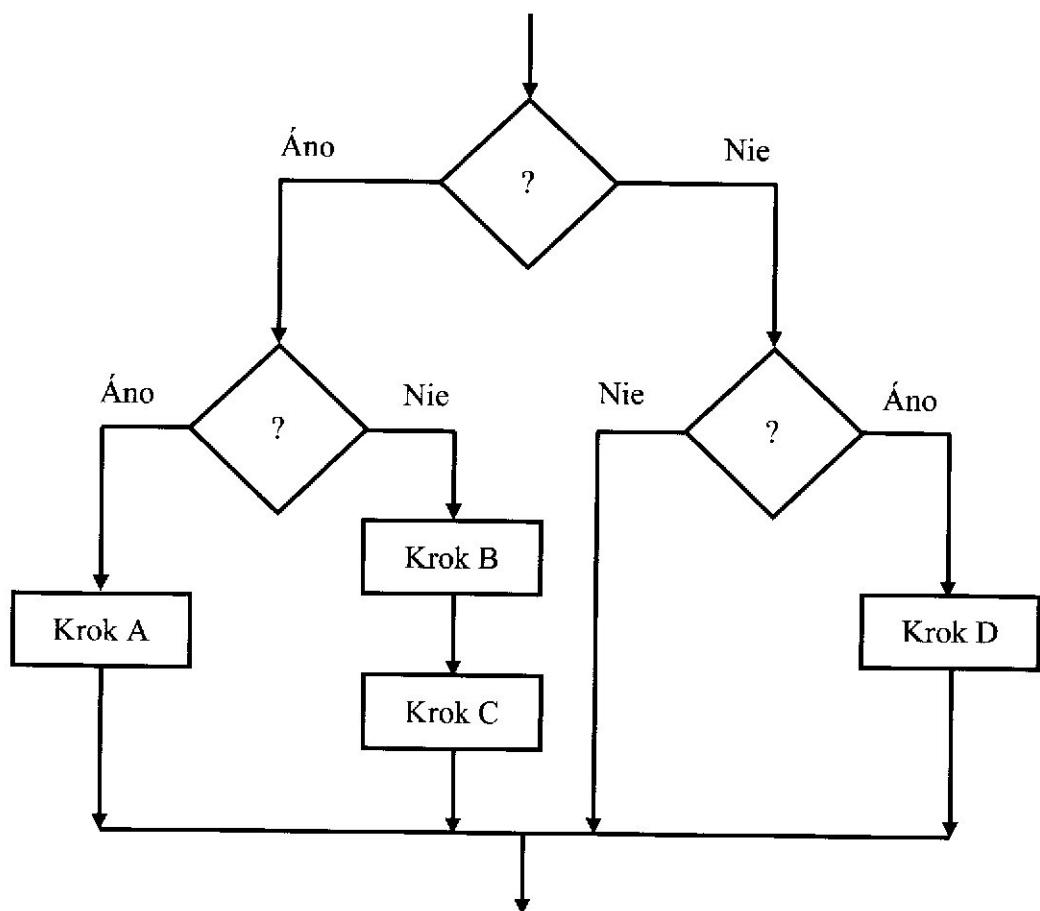
Jednoduché vetvenie úplné znázorňuje obr. 4 a neúplné znázorňuje obr. 5. Viacnásobné vetvenie je znázornené na obr. 6. Pre úplnosť treba dodať, že vo vývojových diagramoch sa zvykne dodržiavať zásada, podľa ktorej musí byť každá podmienka vo vetvení formulovaná ako otázka, na ktorú sa dá odpovedať formou „áno“ alebo „nie“. Z toho vyplýva, že z každej podmienky vychádzajú iba dve vetvy – jedna kladná a jedna záporná. Niektoré iné diagramové techniky na zakreslovanie podnikových procesov (napr. BPMN) však pripúšťajú, aby sa na jednotlivé otázky dalo odpovedať aj inými spôsobmi, než len „áno“ alebo „nie“, a teda pripúšťajú aj existenciu viacerých vetiev vychádzajúcich z jednej podmienky.



Obr. 4: **Jednoduché vetvenie úplné** [Zdroj: autor]



Obr. 5: Jednoduché vetvenie neúplné [Zdroj: autor]



Obr. 6: Príklad viacnásobného vetvenia [Zdroj: autor]

Okrem vetvení sa v podnikových procesoch môžu vyskytovať aj cykly. **Cyklom** nazývame opakované vykonávanie jedného alebo viacerých krokov (t. j. činností) a ako **iteráciu** označujeme jeden prechod cyklom, pri ktorom sa všetky kroky, ktoré sa nachádzajú vo vnútri (t. j. v tele) cyklu, vykonajú iba raz, a to v tom poradí, v akom sú uvedené v tele cyklu. Ak sa teda určitý krok má vykonať presne päťkrát, môžeme to povedať aj tak, že má prebehnúť päť iterácií cyklu obsahujúceho vo svojom vnútri iba tento krok. Pri každom cykle musíme poznáť jasné a jednoznačné podmienky, ktoré určuje, dokedy sa má cyklus vykonávať a kedy sa má jeho vykonávanie skončiť. Cyklus, ktorý by prebiehal donekonečna, nie je prípustný, pretože proces, ktorý by takýto cyklus obsahoval, by sa nikdy neskončil. Podmienka ukončenia cyklu sa môže využiť na vstupe do iterácie, alebo až na výstupe z nej v závislosti od špecifických riešených problémov.

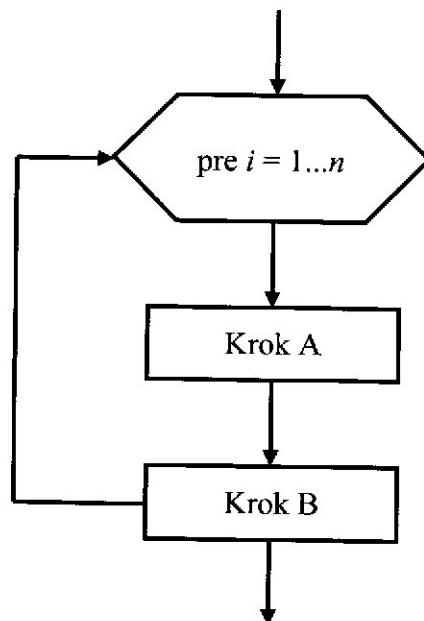
Rozlišujeme tri typy cyklov:

1. **Cyklus s vopred známym počtom opakovaní** – už pred vstupom do prvej iterácie (teda už pred aktivovaním prvého priechodu cyklom) je známe, koľkokrát majú kroky v tele cyklu prebehnúť. Logická podmienka býva stanovená tak, že do ďalšej iterácie cyklu sa vstupuje vtedy, ak dosiaľ neprebehol cielový počet iterácií (obr. 7).
2. **Cyklus s vopred neznámym počtom opakovaní s vyhodnotením logickej podmienky pred vstupom do iterácie** – pred vstupom do prvej iterácie nepoznáme presne celkový počet iterácií, a preto musí byť pred vstupom do každej iterácie vyhodnotená logická podmienka. Ak je táto podmienka splnená, je aktivovaná ďalšia iterácia cyklu. Ak nie je splnená, cyklus sa skončí (ak podmienka nie je splnená už pred vstupom do prvej iterácie, potom cyklus nie je aktivovaný vôbec – kroky v jeho telo neprebehnú ani raz) (obr. 8).
3. **Cyklus s vopred neznámym počtom opakovaní s vyhodnotením logickej podmienky na konci iterácie** – pred vstupom do prvej iterácie nepoznáme presne celkový počet iterácií. Do tela cyklu sa v každom prípade aspoň raz vstúpi a logická podmienka sa vyhodnocuje až na konci každej iterácie. Ak sa logická podmienka splní, je aktivovaná ďalšia iterácia. Ak sa nesplní, cyklus sa skončí (obr. 9).

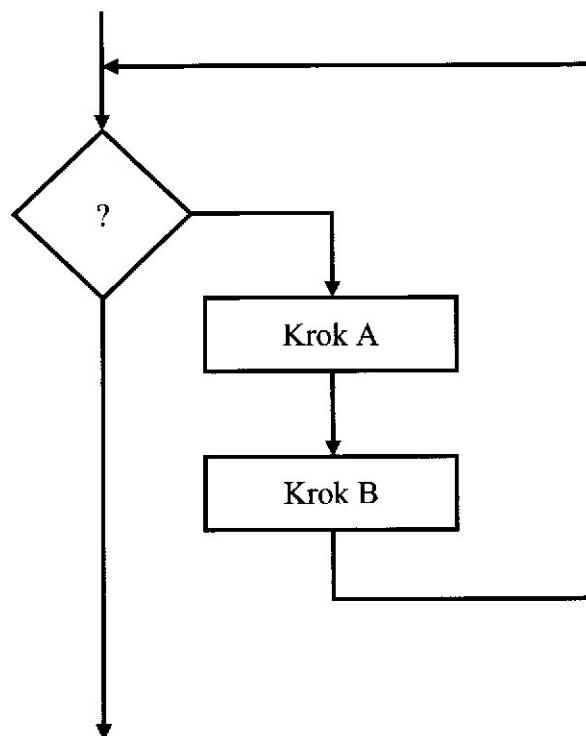
V tele cyklu sa môžu objaviť aj vetvenia či ďalšie cykly. Každý cyklus teda môže obsahovať iné cykly, príp. sám môže byť súčasťou iných cyklov. V takom prípade vzniká hierarchická štruktúra do seba vnorených cyklov. Príkladom vývojového diagramu

obsahujúceho cyklus je diagram na obr. 10, ktorý zobrazuje proces „*Prijímanie nových zamestnancov do zamestnania*“. Proces sa začína presnou špecifikáciou požiadaviek, ktoré by zamestnanci podľa očakávania konkrétneho zamestnávateľa mali splňať vzhľadom na príslušnú pracovnú pozíciu. Keďže zamestnávateľ plánuje uskutočnenie dvoch kôl pohоворov, nasleduje cyklus s vopred známym počtom opakovania. V priebehu oboch iterácií sa potom zrealizujú tri kroky, a to rozoslanie pozvánok na príslušné kolo pohоворu, jeho realizácia a vylúčenie nevyhovujúcich kandidátov. „*Realizácia i-teho kola pohоворu*“ je pomere neurčito formulovaný krok, ktorý by sa dal bližšie konkretizovať v samostatnom vývojovom diagrame. Ten by vyjadroval presný postup, resp. plán konania pohоворu – t. j. všetky jeho kroky v presnom poradí. „*Realizácia i-teho kola pohоворu*“ teda nie je elementárny krok, ale ide skôr o podproces procesu „*Prijímanie nových zamestnancov do zamestnania*“. Ak určitý proces obsahuje jeden alebo viac podprocesov, môžeme hovoriť o tzv. *vertikálnej štruktúre procesu*. Tento pojem bližšie charakterizujeme v ďalších častiach kapitoly 1.

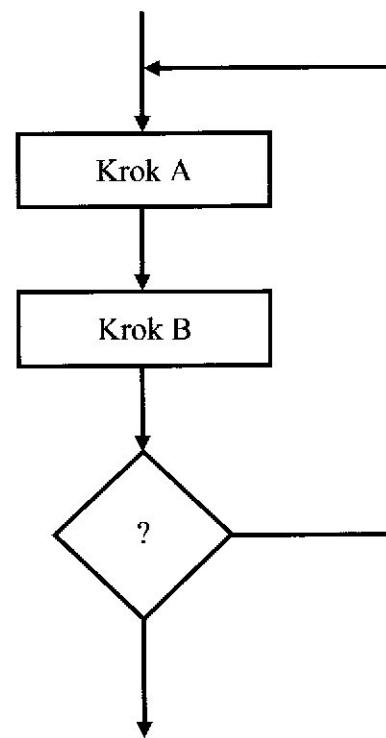
Po realizácii oboch kôl pohоворu nasleduje informovanie úspešných uchádzačov o ich prijatí do zamestnania. Za úspešných absolventov pohоворu pritom považujeme takých, ktorí prešli oboma kolami pohоворu a neboli vylúčení. Záverečným krokom je podpísanie pracovných zmlúv s novoprijatými zamestnancami a ich adekvátné zaškolenie tak, aby mohli začať s napĺňaním svojho poslania.



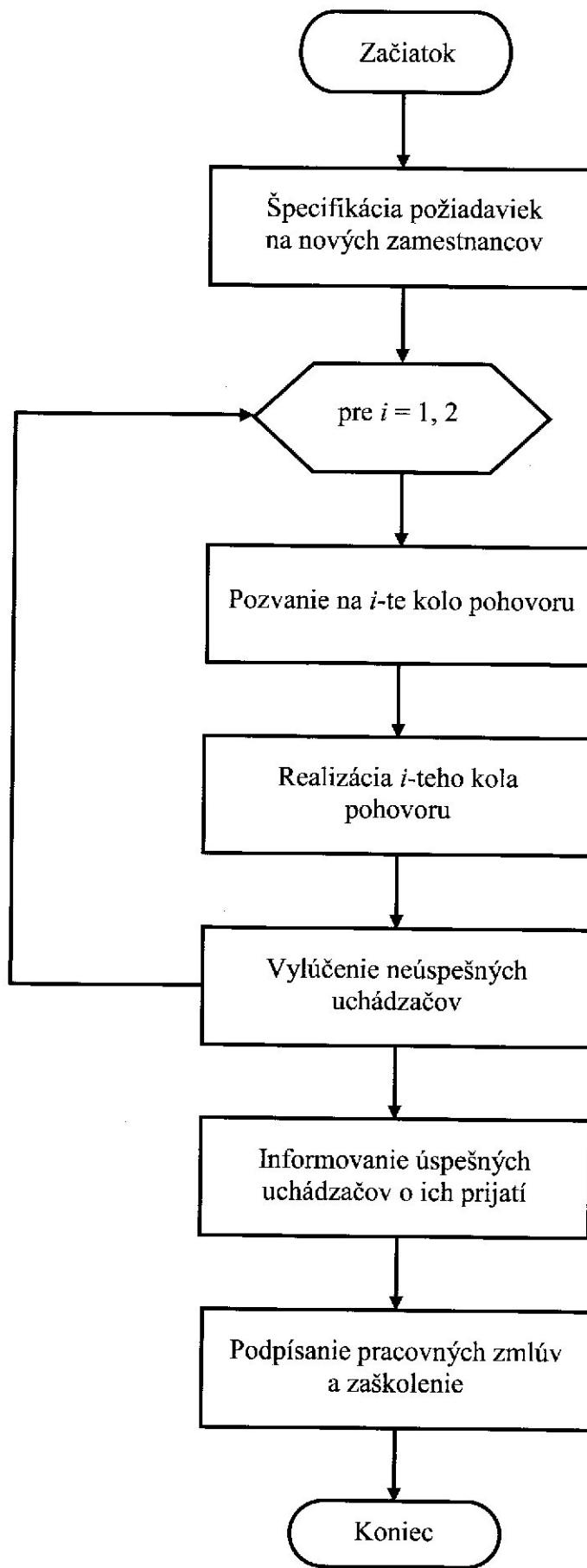
Obr. 7: Cyklus s vopred známym počtom opakovania [Zdroj: autor]



Obr. 8: Cyklus s podmienkou ukončenia vyhodnocovanou pred začatím iterácie  
[Zdroj: autor]



Obr. 9: Cyklus s podmienkou ukončenia vyhodnocovanou na konci iterácie  
[Zdroj: autor]



Obr. 10: Proces „Prijímanie nových zamestnancov do zamestnania“ [Zdroj: autor]

### 1.2.2 BPMN

Ďalšou diagramovou technikou slúžiacou na znázorňovanie priebehu podnikových procesov je BPMN (Business Process Model and Notation). Túto techniku, ktorá v dnešnej dobe predstavuje široko akceptovaný štandard, vyvinula nezisková organizácia BPMI (The Business Process Management Initiative), ktorá sa neskôr zlúčila s konzorcium OMG (Object Management Group) a spoločne pokračujú v jeho rozvíjani.

Diagramy podnikových procesov vytvorené podľa notácie BPMN sú schopné o zobrazovaných procesoch zachytiť viac informácií, než keby sme tieto procesy zakreslili pomocou vývojových diagramov. S tým však súvisí aj jedna nevýhoda: zakresľovanie procesov pomocou BPMN trvá oproti vývojovým diagramom spravidla dlhšie a je komplikovanejšie. Pri rozhodovaní sa pre jednu či druhú z týchto diagramových techník preto treba zvážiť, či je v súvislosti s konkrétnym procesom potrebné uprednostniť množstvo zachytených informácií alebo rýchlosť tvorby diagramu a jeho čitateľnosť (kedže BPMN diagramy môžu byť pre laikov ľahšie pochopiteľné a je náročnejšie sa v nich zorientovať).

Spomedzi mnohých pravidiel zostavovania BPMN diagramov spomeňme tie, ktoré možno považovať za najdôležitejšie:

- Jeden BPMN diagram zobrazuje jeden podnikový proces.
- Každý BPMN diagram je ohraničený rámcem, ktorý sa nazýva „bazén“ (z angl. *pool*).
- Bazén je rozdelený vodorovnými čiarami na niekoľko častí, ktoré sa nazývajú „plavecké dráhy“ (z angl. *swimming lanes*).
- Počet plaveckých dráh v bazéne nie je obmedzený (t. j. bazén môže byť tvorený ľubovoľným počtom plaveckých dráh od 1 do  $N$ ).
- Každá plavecká dráha je priradená jednému aktérovi. Pojem **aktér** označuje osobu, subjekt alebo akúkoľvek jednotku, ktorá má na starosti vykonanie všetkých krokov procesu, ktoré sa nachádzajú v plaveckej dráhe príslušného aktéra.
- Aktérmi podnikových procesov môžu byť najmä:
  - *interné organizačné útvary podniku v napojení na organizačný diagram* (treba uvádzať presný názov príslušného útvaru a jeho číselné označenie presne podľa organizačného diagramu),

- *externé subjekty z podstatného okolia podniku* (ide najmä o obchodných partnerov podniku – dodávateľov, odberateľov, outsourcingové firmy, rozličné agentúry a pod., ale môžu to byť aj banky, úrady, ministerstvá a pod.),
  - *softvér* – ak má určité konkrétné kroky vykonať softvér (napr. rozličné automatizované výpočty, analýzy, zostavovanie reportov a pod.), môžeme mu vytvoriť samostatnú plaveckú dráhu. K názvu softvéru je potom vhodné do zátvorky uviesť názov organizačného útvaru alebo subjektu, ktorý tento softvér obsluhuje, resp. spravuje). Vytvárať samostatné plavecké dráhy pre softvér má zmysel iba v prípade, ak chceme v diagrame odlišovať kroky, ktoré sú automatizované vykonávané určitým softvérom, od neautomatizovaných krovov, ktoré sú vykonávané ručne, resp. bez použitia softvéru.
- Z vyššie uvedeného vyplýva, že BPMN diagram zachytáva nielen postupnosť krovov tvoriačich jeden proces (teda presné poradie ich vykonávania), ale aj to, kto (t. j. ktorý aktér) má na starosti vykonanie príslušného kroku. To znamená, že kroky musia byť vhodne pospájané šípkami, aby bolo jasné poradie ich vykonávania, no zároveň musí byť každý krok zaradený do správnej plaveckej dráhy.
  - Každý proces musí mať jednoznačne vymedzený svoj začiatok a koniec. Tak ako vo vývojových diagramoch aj tu platí, že bod, v ktorom sa proces začína, býva najčastejšie len jeden, ale bodov, v ktorých sa môže končiť, býva často viacero s tým, že tieto body reprezentujú kvalitatívne odlišné konce procesu. Jeden z týchto koncov zvyčajne predstavuje želaný koniec a na ceste k tomuto koncu sa môže vyskytnúť 0 až  $n$  neželaných, predčasných koncov.
  - Vetvenia sa zakresľujú pomocou symbolu tzv. brány. BPMN rozlišuje viacero typov brán, ktoré sú detailne opísané v tab. 3. Na rozdiel od vývojových diagramov sa v BPMN symbol brány nevyskytuje len na začiatku vetvenia, ale musí sa zopakovať aj na jeho konci v bode, kde sa vetvy opäťovne zbiehajú do jednej.
  - Na prechody medzi jednotlivými krokmi procesu sa môžu zakresľovať tzv. *udalosti*. Ak sa medzi krokom A a krokom B nachádza určitá udalosť, znamená to, že na krok B sa nemôže prejsť ihneď po dokončení kroku A, ale musí sa najskôr počkať, až nastane príslušná udalosť. Ide teda o oddialenie prechodu z jedného kroku na druhý. V BPMN existujú rozličné typy udalostí, z ktorých každá má svoj vlastný symbol. Tieto udalosti sú bližšie opísané v tab. 3.

- Všetky kroky procesu musia byť poprepájané orientovanými spojnicami (šípkami), aby bolo jednoznačne stanovené poradie ich vykonávania. Pritom rozlišujeme tzv. *sekvenčné* a *informačné toky*. Sekvenčný tok sa zakresľuje ako plná šípka a vyjadruje iba následnosť krokov (t. j. ich presné poradie). Informačný tok taktiež vyjadruje následnosť krokov, ale okrem toho zdôrazňuje, že medzi aktérm proceusu musí dôjsť k určitej výmene informácií. *Platí zásada, že informačné toky sa používajú len vtedy, ak proces prechádza z jednej plaveckej dráhy do inej, a nikdy sa nesmú vyskytovať len vo vnútri jednej dráhy (vtedy sa používajú sekvenčné toky)*. Informačný tok sa znázorňuje ako šípka s prerušovanou čiarou.

Tab. 3: Prvky BPMN a ich význam (sprac. podľa [OBJ11])

Prvok	Notácia v BPMN	Charakteristika
Udalosť		Vyjadruje situáciu alebo stav, ktorý je významný z hľadiska ďalšieho toku procesu a musí sa naň počkať. Rozlišujeme viaceré typy udalostí.
Počiatočná udalosť		Symbol, ktorý označuje začiatok procesu.
Prechodová udalosť		Ide o udalosť, ktorá nastáva v priebehu procesu – teda niekde medzi jeho začiatkom a koncom (okrem hraníc – začiatku a konca).
Koncová udalosť		Symbol, ktorý označuje ukončenie vykonávania procesu.
Správa		Udalosťou je príchod určitej správy (musí sa na túto správu počkať).
Časovač		Udalosťou je dosiahnutie určitého vopred zadefinovaného časového okamihu alebo uplynutie stanoveného časového intervalu, na ktorý sa musí počkať (dá sa použiť len v spojitosti...).

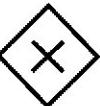
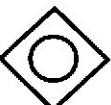
Tab. 3: druhá časť

Prvok	Notácia v BPMN	Charakteristika
Časovač		...s konkrétnym časovým údajom, ktorý musí byť pri tejto udalosti uvedený).
Eskalácia		Ide o udalosť, ktorá môže nastať len vo vnútri podprocesu, a to vtedy, ak do podprocesu zasahuje aktér z vyššej úrovne (teda z procesu, ktorého je príslušný podproces súčasťou).
Podmienená udalosť		Udalosťou je splnenie určitej podmienky alebo pravidla (kým nebude splnené, nesmie sa pokračovať ďalej).
Prepojenie		Udalosťou je presmerovanie toku procesu na iné miesto (prípadne presmerovanie z podprocesu do procesu).
Chyba		Udalosťou je objavenie chyby alebo zaznamenanie neželaného stavu.
Ukončenie		Udalosťou je predčasné ukončenie procesu. Táto udalosť sa používa len pri transakciach.
Kompenzácia		Udalosť, pri ktorej je spustená špeciálna procedúra, resp. podproces, ak nadadený proces čiastočne zlyhá.
Signál		Udalosťou je príchod signálu, ktorý označuje komunikáciu medzi rozličnými procesmi. Symbol signálu môže spustiť iný proces alebo povoliť jeho ukončenie.
Viacnásobný spúšťač		Udalosť, pri ktorej dochádza k spusteniu iného procesu, a to v dôsledku príchodu viacerých signálov, resp. naplnenia viacerých podmienok.

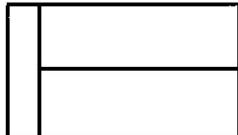
Tab. 3: tretia časť

Prvok	Notácia v BPMN	Charakteristika
Paralelný viacnásobný spúšťač		Inštancia určitého procesu sa nemôže začať, pokračovať alebo byť dokončená, kým nedôjde k paralelnému splneniu viacerých podmienok, resp. k príchodu viacerých signálov.
Ukončenie		Udalosť, ktorá indikuje náhle (predčasné) ukončenie určitého kroku procesu a s ním súvisiacich inštancií procesu.
Úloha		Úloha predstavuje určitú činnosť, ktorá sa musí vykonať v rámci procesu. Ide o aktivitu, ktorá reprezentuje jednotlivé kroky procesu. Úloha býva nerozčleniteľná, teda bud' ju nevieme, alebo nechceme rozčleniť na menšie časti (podúlohy).
Podproces		Podproces je menšia logicky ucelená časť procesu, ktorá sama o sebe predstavuje proces. Jeho priebeh je zakreslený v samostatnom diagrame a pozostáva z menších jednotiek – elementárnych krokov alebo ďalších podprocesov.
Transakcia		Transakcia je špeciálny typ podprocesu. Je to atomická operácia zápisu údajov do databázy, ktorá transformuje databázu z jedného konzistentného stavu do iného konzistentného stavu. Ide o postupnosť krokov pri výmene informácií, ktorá musí byť ako celok považovaná za nedeliteľnú jednotku, aby nedošlo k narušeniu integrity databázy, do ktorej sa tieto informácie zapisujú. Často ide o realizáciu platieb.

Tab. 3: štvrtá časť

Prvok	Notácia v BPMN	Charakteristika
<b>Brána</b>		Brána je symbol, ktorý indikuje, že v toku procesu dochádza k vetveniu. Rozlišujeme viaceré typy brán.
Dátová exkluzívna brána (XOR)		Proces môže pokračovať viacerými vетvami, z ktorých sa môže zvoliť iba jedna, a to na základe splnenia nejakej dátovej podmienky (rozhodovanie podľa údajov súvisiacich s procesom).
Udalostne orientovaná exkluzívna brána (XOR)		Proces môže pokračovať viacerými vетvami, z ktorých sa môže zvoliť iba jedna, a to na základe vzniku či uskutočnenia určitej udalosti (napríklad príchod správy alebo uplynutie časového intervalu).
Paralelná brána (AND)		Proces bude pokračovať viacerými vетvami, pričom všetky budú spustené súčasne. Neskôr by malo dôjsť k opäťovnému zjednoteniu priebehu procesu do jednej vety.
Inkluzívna brána (OR)		Proces môže pokračovať viacerými vетvami, z ktorých sa môže zvoliť jedna veta alebo viacero vetyl (prípadne aj všetky vety). Spustia sa tie vety, pri ktorých je splnená podmienka, ktorá sa s nimi asociouje.
Komplexná brána		Ide o brány, ktoré umožňujú formulovanie zložitejších podmienok vetvenia, a to kombináciou viacerých typov brán.
<b>Sekvenčný tok</b>		Idc o spojnicu vyjadrujúcu sekvenčnú následnosť tokových objektov, ktorými sú úlohy, podprocesy, transakcie, udalosti a brány.

Tab. 3: piata časť

Prvok	Notácia v BPMN	Charakteristika
<b>Informačný tok</b>		Ide o spojnicu vyjadrujúcu komunikáciu medzi dvoma aktérmi procesu. Aktérom procesu môže byť oddelenie, útvar, pracovná pozícia alebo softvér, ktorý má na starosti vykonanie určitej úlohy, resp. úloh v rámci procesu. Môže ísť o ústnu, telefonickú, e-mailovú komunikáciu či ľubovoľný iný prenos údajov.
<b>Asociácia</b>	.....	Spojnica slúžiaca na pripájanie artefaktov k ostatným prvkom (najčastejšie úlohy, brány, a udalosti).
<b>Bazén</b>		Bazén (pool) ohraničuje celý proces a oddeľuje ho od okolitého prostredia.
<b>Plavecké dráhy</b>		Bazén je rozčlenený na plavecké dráhy, pritom každá z nich zodpovedá určitému aktérovi. Všetky ostatné symboly (úlohy, udalosti, spojnice, brány a pod.) sa umiestňujú do týchto dráh. Vďaka tomu je jednoznačne vymedzené, ktorý aktér má na starosti konkrétnu časť, resp. prvok procesu.
<b>Artefakty</b>		Artefakty slúžia na vyjadrovanie doplňujúcich a vysvetľujúcich informácií k ostatným prvkom diagramu. Patria sem anotácie, skupiny a dátové objekty.
<b>Anotácia</b>		Ide o textové komentáre k ostatným prvkom diagramu, ktoré sa k nim pripájajú pomocou asociácie.

Tab. 3: šiesta časť

Prvok	Notácia v BPMN	Charakteristika
Skupiny		Skupiny slúžia na vizuálne zoskupenie úloh a podprocesov, ktoré po logickej stránke patria do jedného celku. Ich zmysel spočíva najmä v sprehľadnení diagramu a v zoskupení niektorých jeho objektov.
Dátové objekty		Tento symbol sa zvykne používať na zdôraznenie toho, že sa na určitom kroku procesu pracuje s databázou alebo sa pristupuje k nejakému dokumentu, príp. že nejaký dokument vzniká ako výsledok príslušného kroku. Symbol dátového objektu sa potom k príslušnému kroku pripojí cez asociáciu (bodkovanú neorientovanú spojnicu).

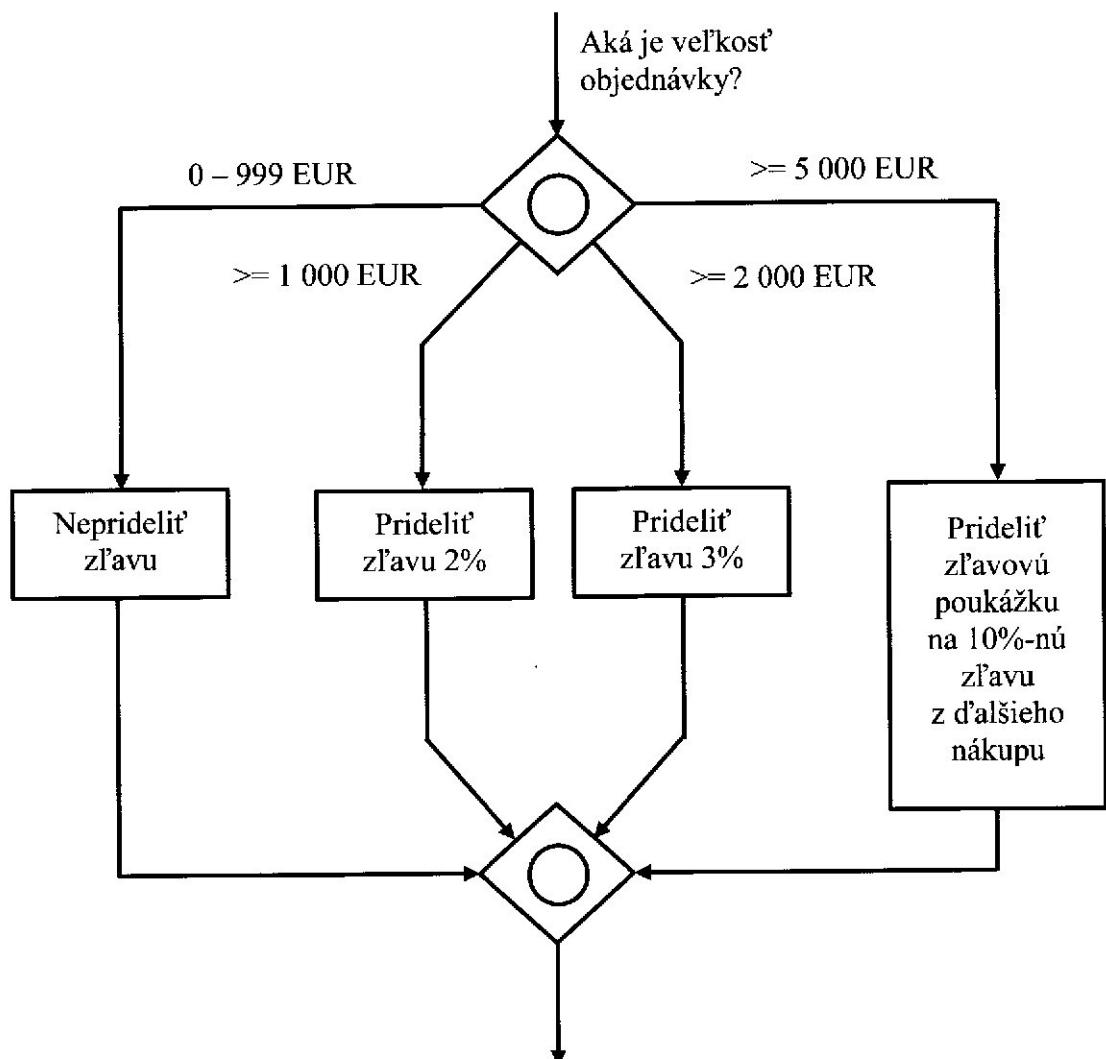
V praxi sa niekedy stretávame s nepochopením, pri akých situáciach je vhodné použiť **inkluzívnu bránu**. Pripomíname, že ide o bránu, z ktorej smú vychádzať viac ako dve vetvy, pričom každá z týchto vetiev sa spája s určitým špecifickým spôsobom naplnenia sledovanej podmienky. Pritom je prípustné, aby sa aktivovali dve alebo viaceré vetvy naraz, ak je táto podmienka pre tieto vetvy splnená.

Predstavme si napr. situáciu, pri ktorej sa firma predávajúca nejaký tovar rozhoduje o pridelení určitého bonusu svojmu zákazníkovi, ktorého výška sa bude odvíjať od veľkosti jeho objednávky (t. j. od jej súhrnejnej finančnej hodnoty). Firma sa rozhodla, že bude v tejto súvislosti rozlišovať nasledovné štyri typy situácií:

- súhrnná hodnota objednávky je nižšia ako 1 000 EUR – zákazník nedostane žiadny bonus,
- súhrnná hodnota objednávky je vyššia alebo rovná 1 000 EUR – zákazník dostane zľavu 2 %,

- súhrnná hodnota objednávky je vyššia alebo rovná 2 000 EUR – zákazník dostane zľavu ďalšie 3 %,
- súhrnná hodnota objednávky je vyššia alebo rovná 5 000 EUR – zákazník dostane navyše (okrem vyššie uvedených zliav) aj zľavovú poukážku na 10%-nú zľavu z ďalšieho nákupu.

Vyššie opísaná rozhodovacia situácia sa dá elegantne znázorniť s použitím inkluzívnej brány tak, ako to zobrazuje obr. 11. Jej použitie však nie je nevyhnutné, keďže sa tá istá situácia dá znázorniť aj za pomoci série na seba nadväzujúcich exkluzívnych brán. Takéto riešenie by ale bolo menej prehľadné a zaberala by v diagrame viac miesta.



Obr. 11: Príklad použitia inkluzívnej brány pri stanovovaní výšky zákazníckeho bonusu  
[Zdroj: autor]

Špecifickým typom brány je aj *paralelná brána*, pri ktorej sa tok procesu vedúci po jednej vetve rozvetví do viacerých vetiev, aby sa neskôr znova spojil do jednej vetvy. Všetky vetvy sa spustia naraz a pred ich opäťovným spojením sa musí počkať, kým nebudú dokončené všetky činnosti vo všetkých vetvách. Každá z vetiev pritom môže obsahovať viacero činností a ich počet ani celková dĺžka trvania v jednotlivých vetvach sa nemusí zhodovať. Symbol paralelnej brány sa používa jednak na začiatku vetvenia, ale aj na jeho konci pri záverečnom spojení vetiev. Ak vetva obsahuje viac ako jeden krok, potom sa všetky jej kroky musia uskutočniť sekvenčne (t. j. jeden po druhom) v tom poradí, v akom sú uvedené.

Paraleлизmus sa často využíva na urýchlenie zložitých výrobných aktivít. Predstavme si výrobu produktu pozostávajúceho z viacerých častí, ktoré sa dajú vyrobiť nezávisle na sebe a na záver sa zmontujú tak, aby dohromady vytvárali jeden celok. Ich výroba sa môže uskutočniť paralelne, vďaka čomu sa tvorba konečného produktu urýchli. Túto situáciu zjednodušene zachytáva obr. 12. Každá z troch paralelných vetiev na tomto obrázku by mohla obsahovať presnú postupnosť krovov smerujúcich k výrobe toho-ktorého dielu (teda dielu A, B alebo C). Jednotlivé diely sa môžu značne odlišovať z hľadiska náročnosti výroby, resp. počtu rozličných činností, ktoré pre ich výrobu treba vykonať, a preto môže každá z trojice paralelných vetiev obsahovať iný počet krovov.

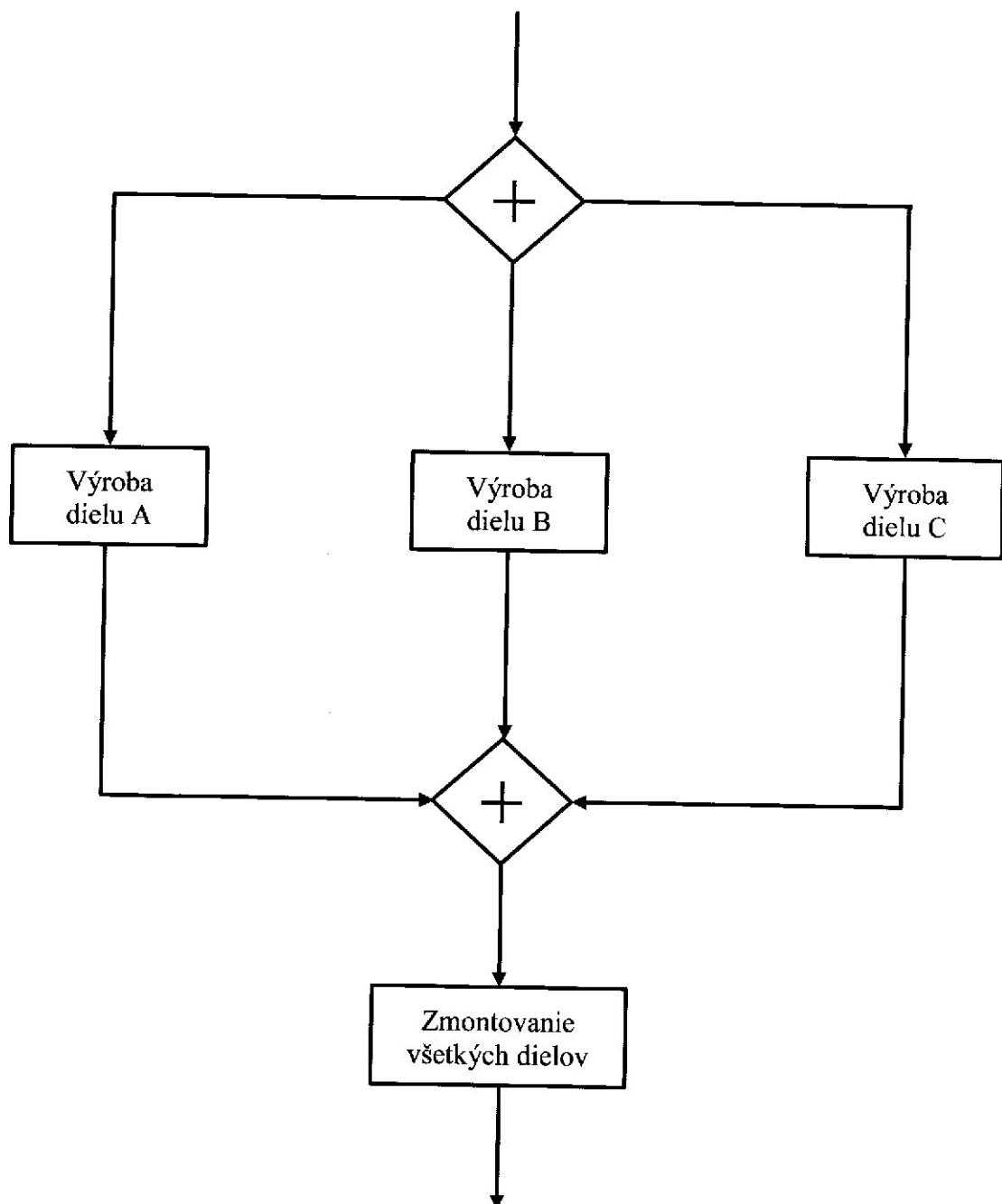
Sekvenčné vykonávanie aktivít má, naproti tomu, zasa zmysel vtedy, ak ide o vzájomne závislé činnosti, pri ktorých záleží na správnom poradí ich vykonávania. Prítom vznikajú reťazce činností, ktoré sa nedajú zrýchliť paraleлизmom.

Použitie štandardu BPMN na znázornenie priebehu podnikového procesu znázorňuje obr. 13. Ide o zjednodušený model procesu „*Výroba posilňovacích strojov*“, do ktorého zasahujú tri typy aktérov:

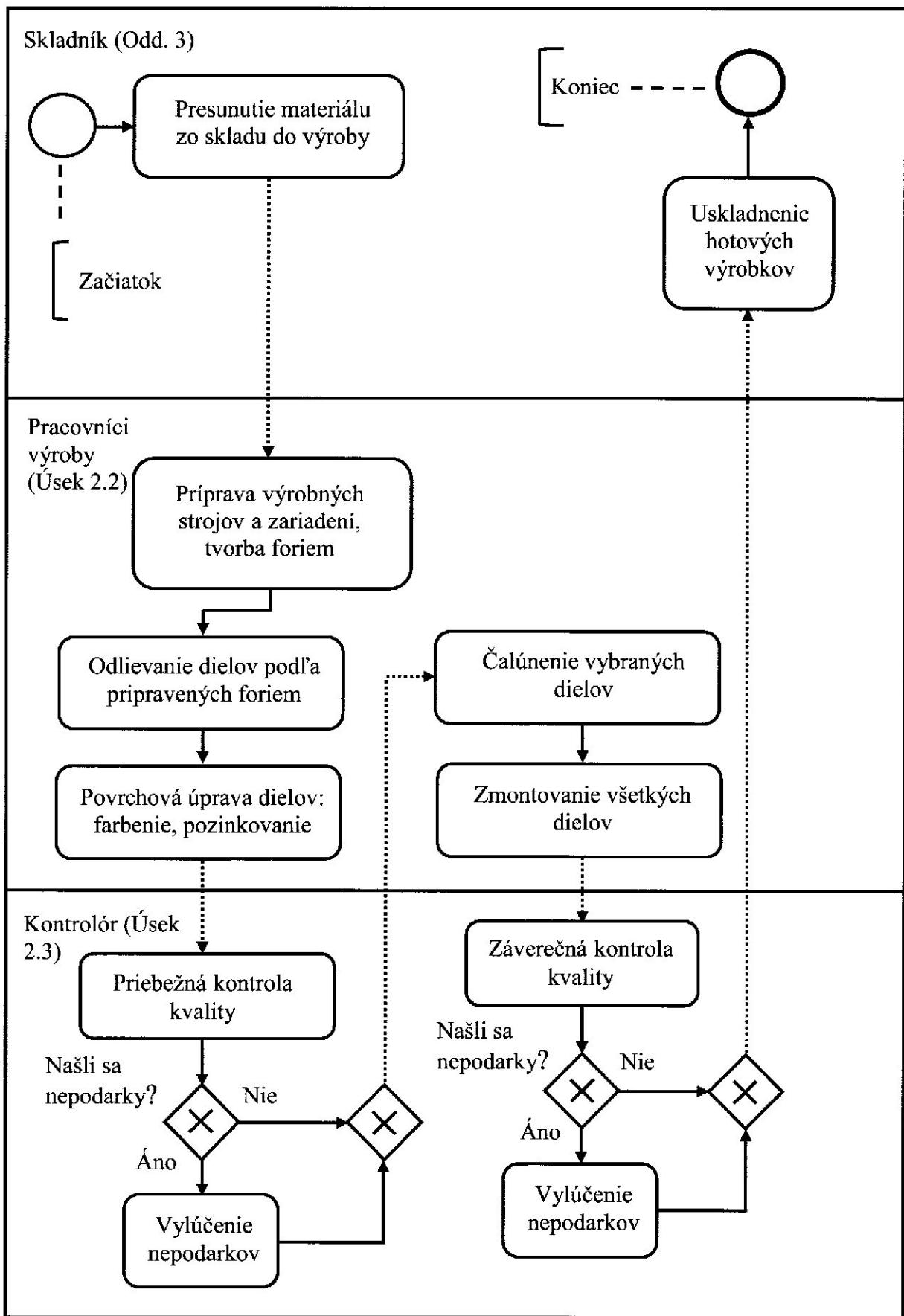
- skladník,
- pracovníci výroby,
- kontrolór.

Pri detailnejšom skúmaní výrobného procesu by sa dal aktér „pracovníci výroby“ rozčleniť na viacerých bližšie špecifikovaných aktérov – vzhľadom na konkrétnu úlohu, ktorú v tomto procese plnia (napr. špecialisti na odlievanie dielov, špecialisti na povrchovú úpravu, montéri a pod.). Rozmiestnením jednotlivých krovov procesu do plaveckých dráh potom jednoznačne vymedzujeme, ktorý aktér má na starosti ktorý krok procesu. Takto sa dá vidieť aj

vyťaženosť jednotlivých aktérov, t. j. miera ich zapojenia do toho-ktorého procesu. Začiatočná značka sa dáva do plaveckej dráhy toho aktéra, u ktorého sa proces začína (t. j. dáva sa k aktérovi, ktorý vykonáva prvý krok procesu), a koncová značka sa zasa dáva do dráhy toho aktéra, u ktorého sa proces končí (t. j. tento aktér vykonáva jeho posledný krok). V príklade na obr. 13 sa proces zhodou okolností začína aj končí u toho istého aktéra, a tým je skladník.



Obr. 12: Príklad zjednodušene znázorňujúci paralelizmus vo výrobe [Zdroj: autor]



Obr. 13: Diagram procesu „Výroba posilňovacích strojov“ [Zdroj: autor]

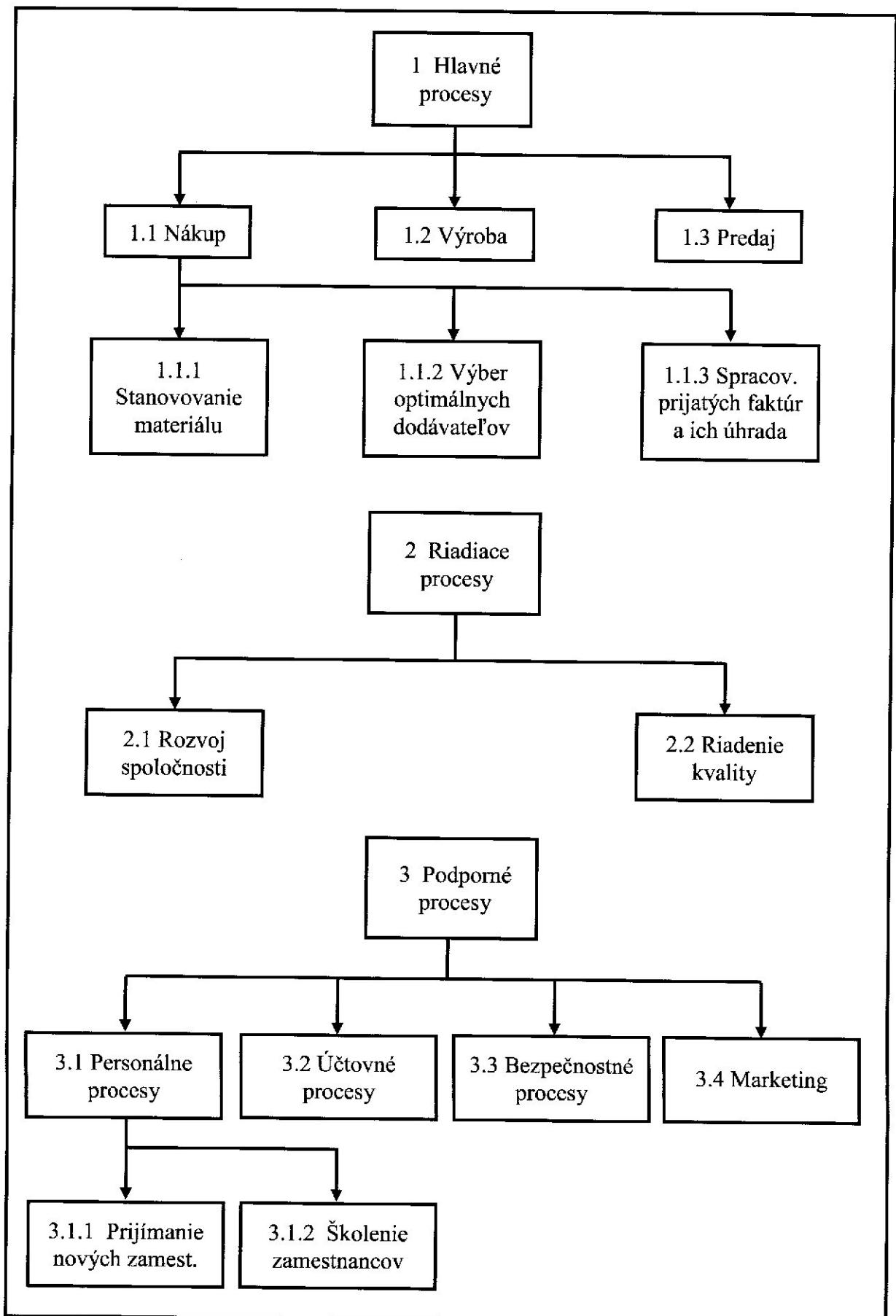
### 1.2.3 Procesné mapy

Ako sme už spomenuli, vývojové diagramy a BPMN diagramy sú vhodné diagramové techniky na znázorňovanie priebehu podnikových procesov, t. j. na zachytenie postupnosti krokov, z ktorých pozostávajú. Užitočnou diagramovou technikou súvisiacou s podnikovými procesmi je však aj **procesná mapa**. Jej cieľom nie je znázorňovať priebeh určitého podnikového procesu, ale predstavuje hierarchicky štruktúrovany zoznam všetkých podnikových procesov identifikovaných v určitom podniku, vrátane podprocesov, z ktorých pozostávajú. Zložité podnikové procesy s dlhým a komplikovaným priebehom sa totiž často za účelom ich zjednodušenia a sprehľadnenia rozkladajú na podprocesy. *Podproces je menší podnikový proces, ktorý je súčasťou väčšieho podnikového procesu.*

Príkladom takéhoto hierarchického rozkladu procesu na podprocesy môže byť napr. proces „*Nákup*“. Ide o pomerne komplikovaný proces, v ktorom môžeme identifikovať viaceré zmysluplné podprocesy, ako napr. „*Stanovovanie materiálu na nákup a potrebného množstva*“, „*Výber optimálnych dodávateľov*“, či „*Spracovanie a úhrada prijatých faktúr*“. Je evidentné, že nejde o elementárne činnosti, ale o postupnosti viacerých krokov, pričom každá takáto postupnosť predstavuje ucelený podnikový proces s konkrétnym cieľom. Ide teda o plnohodnotné podnikové procesy, ktoré sú súčasťou širšieho procesu s názvom „*Nákup*“ a vo vzťahu k nemu ide o jeho podprocesy. Sú teda na hierarchicky nižšej úrovni.

Procesná mapa okrem toho rozdeľuje procesy na *hlavné, riadiace a podporné*. Každý proces musí byť zaradený do jednej z týchto kategórií a následne sa v prípade potreby môže členiť na podprocesy. Rozdiel medzi hlavnými, riadiacimi a podpornými procesmi sme už spomínali v kapitole 1.2. Každá z týchto kategórii má v procesnej mape unikátné číslo (1, 2 alebo 3) a od neho sa odvíjajú čísla jednotlivých procesov a podprocesov v tej-ktorej kategórii.

Príklad procesnej mapy znázorňuje obr. 14. Hierarchickú dekompozíciu procesu na podprocesy (t. j. tzv. *vertikálnu štruktúru procesu*) demonštruje proces *1.1 Nákup*. Tento proces obsahuje tri podprocesy, a to *1.1.1 Stanovovanie množstva materiálu, ktorý treba nakúpiť*; *1.1.2 Výber optimálnych dodávateľov* a *1.1.3 Spracovanie prijatých faktúr a ich úhrada*. Na tomto príklade si môžeme tiež všimnúť zaradenie dvoch čiastkových procesov, ktorími sú proces *3.1.1 Prijímanie nových zamestnancov* a *3.1.2 Školenie zamestnancov*, do jednej súhrnejnej kategórie, a to *3.1 Personálne procesy*, ktorá je podkategóriou kategórie *3 Podporné procesy*.



Obr. 14: Procesná mapa [Zdroj: autor]

### **1.3 Porovnanie funkčného a procesného prístupu k riadeniu podniku**

V predchádzajúcich podkapitolách kapitoly 1 sme objasnili, čo je to organizačná štruktúra podniku, čo sú to funkcie podnikových útvarov a čo sú to podnikové procesy. Z tohto hľadiska existujú dva základné prístupy k riadeniu podniku: *funkčný* (alebo *funkcionálny*) a *procesný prístup*. Historicky starší je **funkčný prístup**, ktorý nahliada na podnik iba ako na množinu hierarchicky usporiadaných funkcií. Ako uvádza Bruckner a kol.: „*pri funkčnom pohľade členime podnik ako celok na niekoľko funkčných oblastí*“ (napr. nákup, výroba, predaj, personalistika, účtovníctvo, správa financií, bezpečnosť a pod.) [BRU12]. Každú z týchto oblastí môžeme opäť rozčleniť na čiastkové funkčné oblasti a tak ďalej, až sa dostaneme k funkciám, resp. činnostiam, ktoré už nemá zmysel ďalej rozčleňovať [BRU12]. Toto členenie sa vykonáva v nadväznosti na organizačnú štruktúru, t. j. pre každú funkciu musí byť jasne stanovené, ktorý organizačný útvar z organizačného diagramu má na starosti jej vykonanie (príp. jej kontrolu alebo riadenie – ako sme spomínali v podkapitole 1.1 v časti, kde sme rozoberali relačnú maticu). Ako uvádzajú Basl a Blažiček: „*Paradigma funkčného manažmentu bola po prvýkrát definovaná Adamom Smithom v jeho knihe Pojednanie o podstate a pôvode bohatstva národov z roku 1776. Táto paradiigma hovorila, že prácu treba rozdeliť na také malé úkony, aby ich zvládol každý pracovník – aj nekvalifikovaný a nevzdelený.*“ [BAS12]. Basl a Blažiček ďalej dodávajú, že tento prístup ako jeden z prvých uplatnil napr. Henry Ford pri výrobe automobilov, a to tak, že u montážnej linky nechal každého pracovníka, aby na automobile namontoval iba jedinú časť (t. j. každý pracovník bol úzko špecializovaný iba na jednu konkrétnu činnosť, ktorá sa rutinne opakovala stále dookola, čím sa znižovala pravdepodobnosť, že niečo pokazí) [BAS12].

Je zrejmé, že pri tomto spôsobe prerozdeľovania pracovných povinností vzniká medzi zamestnancami množstvo hierarchických väzieb (t. j. vzťahov nadriadenosti a podriadenosti) – treba veľa pracovníkov, ktorí vykonávajú úzko špecializované činnosti, a treba dôsledne kontrolovať výkon práce každého jedného z nich. Funkčne riadené podniky teda zvyknú mať vysoký stupeň vertikálneho členenia organizačnej štruktúry – býva v nich množstvo hierarchických úrovní.

**Výhodou funkčného prístupu** k riadeniu je dokonalý prehľad o náplni práce a kompetenciach jednotlivých zložiek organizačnej štruktúry podniku a tiež prehľad o hierarchickej štruktúre jednotlivých funkčných oblastí. **Nevýhodou funkčného prístupu** je zasa to, že na jednotlivé funkčné oblasti sa nenahliada v ich vzájomnej nadväznosti, ale izolovane. Stráca sa tak prehľad o tom, ako na seba jednotlivé funkcie nadväzujú, kto s kým

v podniku spolupracuje a čo presne má byť výsledkom ich spolupráce. Manažment funkčne riadeného podniku sa teda nezaoberá tým, aké podnikové procesy v ňom prebiehajú, čo má byť výstupom ktorého procesu a pre koho je tento výstup určený<sup>1</sup> a ani tým, ktorá funkcia je súčasťou ktorého procesu a akú má v ňom pozíciu vzhladom na jeho celkový priebeh. Pri veľkých podnikoch s množstvom organizačných útvarov sa dokonca môže stávať, že dva rôzne útvary vykonávajú duplicitne tú istú činnosť a ani o tom nevedia (napr. dva rôzne útvary vykonávajú určitú analýzu, ktorá sa po obsahovej stránke na 70% zhoduje, a teda tieto útvary duplicitne zhromažďujú a vyhodnocujú podkladové údaje pre túto analýzu). *Každý organizačný útvar je pri funkčnom riadení hodnotený za to, či vykonáva jemu zverené funkcie v súlade s očakávaniami manažmentu.* Kvalita spoluprácc s inými útvarmi, ktoré sa podieľajú na vytváraní toho istého výstupu pre toho istého zákazníka, sa pritom nezohľadňuje. Pri funkčnom riadení tak často dochádza k zbytočným prieťahom pri prechodoch z jednej funkcie na druhú (t. j. keď jeden útvar „odovzdáva štafetu“ druhému útvaru).

**Procesný prístup** k riadeniu podniku sa snaží odstraňovať vyššie uvedené nedostatky funkčného prístupu. Jeho cieľom je identifikovať všetky procesy, ktoré v skúmanom podniku prebiehajú, alebo by v ňom mali prebiehať, stanoviť pravidlá, ktorými by sa mal riadiť ich priebeh, a pre každý proces stanoviť konkrétnu osobu (t. j. vlastníka procesu) tak, aby táto osoba dbala na dosahovanie cieľov procesu. Bruckner a kol. neodporúčajú začínať procesnú analýzu zostavovaním zoznamu všetkých procesov a až následne sa venovať analýze a modelovaniu správneho priebehu jednotlivých procesov, ale odporúčajú presne opačný postup. Procesy teda treba najskôr rozpoznať (identifikovať), namodelovať ich správny priebeh a až na záver zostaviť zoznam všetkých procesov. Vhodnou diagramovou technikou na reprezentáciu takéhoto zoznamu je procesná mapa, o ktorej sme pojednávali v kapitole 1.2.3 [BRU12]. Ako sme už spomenuli, procesná mapa rozčleňuje podnikové procesy na hlavné, riadiace a podporné, pričom procesy zaradené do týchto kategórií sa môžu členiť na podprocesy, tie na ďalšie podprocesy atď. *Procesná mapa teda zachytáva hierarchickú dekompozíciu procesov na podprocesy, no bez uvedenia bližších detailov týkajúcich sa presnej pozície podprocesov v nadradenom procese.*

---

<sup>1</sup> Výstup procesu môže byť určený pre externého alebo interného zákazníka procesu. Externým zákazníkom môže byť externý subjekt (napr. firma, inštitúcia, štát) alebo fyzická osoba – spotrebiteľ (t. j. výstup procesu je určený „pre niekoho zvonku“), zatiaľ čo interným zákazníkom môže byť niektoré oddelenie, útvar či konkrétna osoba v rámci organizačnej štruktúry toho istého podniku, v ktorom tento proces prebieha (t. j. výstup procesu je určený pre vnútorné potreby podniku). Ak je výstup procesu určený pre interného zákazníka, potom môže (ale nemusí) ísť o situáciu, pri ktorej je dokončenie jedného procesu spúšťacou udalosťou pre iný proces. V takýchto situáciách dochádza k tzv. zreteženiu procesov – jeden proces priamo nadvázuje na iný, preberá jeho výstupy a začína sa tam, kde sa predchádzajúci proces skončil.

V rámci procesnej analýzy sa však treba zaoberať aj priamymi vzťahmi a vzájomnými nadväznosťami procesov. *Výstupy jedného procesu môžu byť totiž vstupmi do iného*, a to nielen na jeho začiatku, ale aj v jeho priebehu (t. j. jeden proces sa dostane do bodu, keď na to, aby mohol pokračovať, potrebuje dostať výstup iného procesu – ten sa musí spustiť, poskytnúť svoj výstup procesu, ktorý ho zavola, t. j. inicioval jeho spustenie, a následne môže pôvodný proces pokračovať). *Môžu tak vznikať určité zreteženia na seba nadväzujúcich procesov*, ktoré sú podprocesmi širšieho (t. j. nadradeného) procesu a dohromady sa v správnom poradí spolu-podieľajú na zabezpečovaní jeho výstupov.

Rozlišujeme teda vertikálnu a horizontálnu štruktúru procesov. **Vertikálna štruktúra** hovorí, že určitý proces sa vnútorene člení na podprocesy a tie sa môžu opäť členiť na podprocesy, čím vzniká hierarchia niekoľkých úrovni procesov a podprocesov. Takúto skladbu procesu vhodným spôsobom zachytáva už spomínaná procesná mapa, ale slúžia na to aj iné diagramové techniky, napr. *diagram procesných vlákien* (process thread diagram) a *Juríkov-Schmidtov diagram* (označovaný aj ako *diagram kompozície zloženého procesu*; v angl. Composition Diagram of a Complex Process). Tieto diagramové techniky umožňujú znázorniť vertikálnu štruktúru procesu iba v jedinom diagrame. Vynikajúcimi diagramovými technikami na znázornenie **horizontálnej štruktúry procesu** (teda postupnosti krokov, ktoré sú na tej istej hierarchickej úrovni) sú vývojový diagram a BPMN diagram. Pre úplnosť musíme dodať, že obe tieto diagramové techniky sa sice dajú použiť aj na znázornenie vertikálnej štruktúry procesu, no iba vo forme viacerých prepojených diagramov.

Proces, ktorý sa dá hierarchicky rozčleniť na podprocesy (resp. ich môžeme nazvať aj podriadené procesy), sa označuje ako **zložený** (t. j. komplexný) **proces**. Proces, ktorý už nemožno ďalej členiť na podriadené procesy alebo takéto členenie nemá praktický význam, sa zasa označuje ako **elementárny proces**.

Pre každý identifikovaný proces je potrebné:

- stanoviť jeho vlastníka,
- identifikovať zákazníka procesu (interného alebo externého),
- stanoviť ciele procesu tak, aby boli jednoznačné a merateľné,
- určiť výstupy, ktoré má proces poskytnúť svojmu zákazníkovi,
- určiť všetky vstupy, ktoré tento proces potrebuje na to, aby mohol poskytovať očakávané výstupy,

- stanoviť presnú postupnosť krokov správneho priebehu procesu a znázorniť ju pomocou niektornej z vhodných diagramových techník (napr. vývojové diagramy, BPMN diagramy) alebo tabuľkových techník (napr. RACI matice, rozhodovacie tabuľky)<sup>2</sup>,
- identifikovať všetkých aktérov (interných aj externých) participujúcich na realizácii krokov, z ktorých proces pozostáva,
- identifikovať všetky regulátory riadenia (t. j. smernice, nariadenia, zákony, vyhlášky a iné dokumenty, ktoré musí tento proces rešpektovať),
- identifikovať všetky slabé miesta procesu, t. j. všetky jeho segmenty, na ktorých sa proces môže skončiť neúspechom,
- pravidelne vyhodnocovať efektívnosť vykonávania procesu (t. j. mieru, v akej dochádza k napĺňaniu cieľov procesu), čo má na starosti jeho vlastník. Ak jeho efektívnosť nenapĺňa očakávania, potom je potrebné hľadať príčiny a snažiť sa zodpovedajúcim spôsobom upraviť pravidlá jeho priebehu.

*Základnou myšlienkovou procesného prístupu k riadeniu podniku je snaha o to, aby každý proces prinášal čo najväčšiu hodnotu pre svojho zákaznika. Kontrola sa teda nezameriava iba na to, či si jednotlivé organizačné útvary, ktoré participujú na procese, individuálne splnili svoje povinnosti, ale či proces ako celok prináša pre svojho zákazníka takú hodnotu, ktorú tento zákazník potrebuje a očakáva. To je základný rozdiel medzi funkčným a procesným prístupom k riadeniu.*

Pri procesnom riadení nesmie žiadny proces zostať osirotený, t. j. nesmie nastať situácia, že by za neho neboli nikto zodpovedný, a to či už ako vlastník, prevádzkovateľ alebo vykonávateľ. Truneček vo svojej publikácii *Management v informační společnosti* vymenúva 4 významné znaky, ktorými sa procesný prístup k riadeniu odlišuje od funkčného [TRU99]:

1. riadenie aj organizačná štruktúra sú skôr horizontálne než vertikálne. Organizačné útvary majú vysoký stupeň autonómie a sú medzi nimi skôr horizontálne väzby, ktoré vyjadrujú rovnocennosť, než vertikálne, ktoré vyjadrujú nadriadenosť a podriadenosť. Vertikálne väzby sa vyskytujú najmä pri klúčových celopodnikových strategických a rozvojových činnostach.

---

<sup>2</sup> Viac o tabuľkových technikách určených na znázorňovanie podnikových procesov alebo ich časti sa dá dočítať v [JUR18]. Rozhodovacie tabuľky pritom slúžia skôr na sprehľadnenie zložitých rozhodovacích situácií, ktoré obsahujú viacnásobné vetvenia, než na zakreslovanie celých podnikových procesov. RACI matice sú vhodné na znázornenie postupnosti krokov, ktoré tvoria určitý proces, ako aj na zachytenie informácií o tom, ktorí aktéri sa podieľajú na jednotlivých krokoch. Nchodia sa však na znázornenie zložitejších vetvení, cyklov a paralelizmu.

2. Tímy, ktoré zabezpečujú jednotlivé podnikové procesy, môžu pracovať natoľko samostatne, až sa môže zdať, že sami podnikajú vo vnútri podniku (túto zásadu sformuloval už Tomáš Baťa v 30. rokoch 20. stor.). Inými slovami, každý tím, ktorý zabezpečuje jeden proces, by sa mal snažiť o to, aby príslušný proces ako celok dosahoval čo možno najlepšie výsledky.
3. Preferuje sa kolektívna motivácia pred motiváciou jednotlivca. Motivácia sa teda viaže na dosahované výsledky procesu – ak sú tieto výsledky dobré, odmenený bude celý tím. Vďaka tomu sú jednotliví členovia tímu motivovaní, aby sa usilovali o čo najefektívnejšiu vzájomnú spoluprácu a aby potláčali rôzne antipatie (t. j. negatívne postoje jedného voči druhému), ktoré by zbytočne predlžovali dokončovanie procesu alebo by viedli k jeho väčšej chybovosti. Zákazník procesu a jeho spokojnosť s výsledkami je totiž hlavným meradlom výšky odmeny pre príslušný tím.
4. Vedenie tímu je založené skôr na koučovaní než na autoritatívnom prikazovaní. Mení sa teda úloha manažérov tímov (t. j. vlastníkov jednotlivých procesov), ktorí by zamestnancov mali skôr motivovať a mali by im umožniť pracovať čo najviac samostatne.

Podrobnejšiu analýzu rozdielov medzi funkčným a procesným riadením uvádzajú Kovář, Kožíšková a Hrazdilová-Bočková v knihe *Teorie průmyslových podnikatelských systémů II* [KOV04]. Na výsledkoch tejto analýzy je postavená tab. 4.

Tab. 4: Porovnanie funkčného a procesného riadenia (sprac. podľa [KOV04])

Charakteristika	Funkčné riadenie	Procesné riadenie
Základný princíp	Deľba práce	Prepájanie činností
Základná stavebná jednotka	Funkcia	Proces ako postupnosť čiastkových činností
Pozornosť sa sústredí na	Správne vykonanie každej čiastkovej činnosti	Výsledok
Charakter výroby	Hromadná	Variantnosť – vlastník vie dohliadnuť na prispôsobenie...

Tab. 4: druhá časť

Charakteristika	Funkčné riadenie	Procesné riadenie
Charakter výroby		...konkrétnej inštancie procesu potrebám konkrétneho zákazníka
Základné aktívum	Kapitál	Znalosti
Predpoklad úspechu	Objem, rýchlosť	Pružnosť
Podnik ako systém	Koordinácia oddelených prvkov	Snaha o synergický efekt
Ukazovatele úspešnosti	Ekonomicke ukazovatele	Pridaná hodnota pre zákazníka
Organizačná štruktúra	Strmá pyramída	Horizontálna, plochá
Riadenie	Hierarchické	Laterálne (naprieč útvarmi)
Právomoci, zodpovednosť	Za konkrétnu činnosť (funkciu) alebo útvar	Za proces ako celok, ale aj za čiastkové činnosti
Vzťah k podriadeným	Kontrola, prikazovanie, koordinácia, tvrdé prvky	Splnomocňovanie, koučovanie, mäkké prvky
Ukazovatele podniku	Ekonomická analýza	Analýza procesov
Odstraňovanie problémov	Riešenie dôsledkov	Hľadanie príčin
Manažment riadi	Jednotlivcov	Tímy
Vnútropodnikové prostredie	Vzájomná konkurencia medzi vnútropodnikovými útvarmi	Spolupráca
Charakter práce	Špecializácia – rozdelenie práce na rozsahovo čo najmenšie činnosti	Prepájanie činností
Kvalifikácia	Nenáročná, stačia užšie znalosti	Náročné na kvalifikáciu, treba širšie znalosti
Motivácia	Splnenie ukazovateľov spojených s funkciou	Vyhodnotenie hodnoty, ktorú proces prináša pre zákazníka
Komunikácia	Lineárne vertikálna	Viac horizontálna

## Zhrnutie kapitoly 1

V kapitole 1 sme sa oboznámili so základnými princípmi funkčného, a rovnako tak procesného prístupu k riadeniu podniku. Pri oboch týchto prístupoch je dôležité vychádzať z organizačnej štruktúry podniku, ktorá určuje, z akých organizačných útvarov pozostáva ten-ktorý podnik a aké sú medzi nimi väzby. Tie môžu byť buď horizontálne (rovnocennosť útvarov), alebo vertikálne (nadriadenosť a podriadenosť útvarov). Každý organizačný útvar musí zabezpečovať aspoň jednu alebo viacero funkcií. Funkcia reprezentuje činnosť, ktorú tento útvar na rutinnej báze vykonáva, resp. zabezpečuje. Je to teda poslanie určitého konkrétneho útvaru (t. j. náplň jeho práce). Podnikový proces predstavuje zaužívaný spôsob, ako sa v konkrétnom podniku rieši určitý konkrétny problém, resp. situácia, ktorá v ňom opakovane nastáva. Je to teda presná postupnosť krokov, ktorá smeruje k vyriešeniu tejto situácie a na ktorú sa príslušný podnik môže spoloahnúť vždy, keď táto situácia nastane. Za vykonanie jednotlivých krokov procesu musí niekto zodpovedať, čo značí, že každý krok procesu je vlastne funkciou prináležiacou niektorému organizačnému útvaru alebo konkrétej osobe, resp. pracovnej pozícii. Na modelovanie organizačnej štruktúry podniku sa používa organizačný diagram, v ktorom sú znázornené všetky organizačné útvary. Hierarchický diagram funkcií je zasa zoznam všetkých funkcií, ktoré prináležia organizačným útvarom z organizačného diagramu. Vzájomnú nadväznosť medzi týmito diagramami zachytáva relačná matica. Diagram funkčných závislostí zachytáva predovšetkým hmotné, finančné a informačné toky medzi zväčša rovnocennými funkciami z hierarchického diagramu funkcií. Pri modelovaní podnikových procesov rozlišujeme ich vertikálnu a horizontálnu štruktúru. Vertikálna štruktúra zachytáva hierarchický rozpad procesu na podprocesy, ktorý môže mať viacero úrovní. Najjednoduchším diagramom na znázornenie vertikálnej štruktúry procesov je procesná mapa, ktorá je zároveň zoznamom všetkých procesov prebiehajúcich v podniku. Horizontálna štruktúra procesu vyjadruje presné poradie menších aktivít, z ktorých proces pozostáva. Môže obsahovať lineárne postupnosti krokov (teda jeden krok za druhým), ale aj vetvenia, cykly či paraleлизmus. Na znázornenie horizontálnej štruktúry procesu je vhodné použiť vývojové diagramy alebo BPMN diagramy. BPMN diagramy majú oproti vývojovým diagramom bohatšiu notáciu (viac symbolov so špecifickým významom), no sú komplikovanejšie na zakreslovanie aj porozumenie. Existujú však aj tabuľkové techniky, ktoré sa dajú využiť pri znázorňovaní horizontálnej štruktúry procesu, ktorými sú najmä RACI matice alebo rozhodovacie tabuľky. Výsledkom tabuľkových techník nie je diagram, ale tabuľka. Viac o nich sa dá dočítať napr. v [JUR18].

## Literatúra ku kapitole 1

- [BAS12] BASL, J. – BLAŽÍČEK, R.: *Podnikové informační systémy*. 3. vyd. Praha: Grada Publishing, a. s., 2012. 328 s. ISBN 978-80-247-4307-3.
- [BRU12] BRUCKNER, T. – VOŘÍŠEK, J. – BUCHALCEVOVÁ, A. – STANOVSKÁ, I. – CHLAPEK, D. – ŘEPA, V.: *Tvorba informačních systémů*. Praha: Grada Publishing, a. s., 2012. 360 s. ISBN 978-80-247-4153-6.
- [DAV90] DAVENPORT, T. H. – SHORT, J.: The New Industrial Engineering. Information Technology and Business Process Redesign. In: *Sloan Management Review* [online], Vol. 31, No. 4, s. 11 – 27. [Cit. 10. 7. 2016]. Dostupné na internete: <<http://dspace.mit.edu/bitstream/handle/1721.1/48613/newindustrialeng00dave.pdf>>.
- [ISO85] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION: ISO 5807:1985. In: *iso.org* [online]. [Cit. 10. 3. 2020]. Dostupné na internete: <[http://www.iso.org/iso/catalogue\\_detail.htm?csnumber=11955](http://www.iso.org/iso/catalogue_detail.htm?csnumber=11955)>.
- [JUR18] JURÍK, P.: *Informačné systémy v podnikovej praxi*. 2018. 2. vyd. Nové Zámky: Tlačiareň MERKUR, s. r. o. 186 s. 978-80-970233-7-9.
- [KAI13] KAIZEN INSTITUTE: Key role of EMPLOYEES in Business Process Improvement. 2013. In: *in kaizen.com* [online]. [Cit. 7. 7. 2020]. Dostupné na internete: <<https://in.kaizen.com/blog/post/2013/05/31/key-role-of-employees-in-business-process-improvement.html>>.
- [KOV04] KOVÁŘ, F. – KOŽÍŠKOVÁ, H. – HRAZDILOVÁ-BOČKOVÁ, K.: *Teorie průmyslových podnikatelských systémů II*. 2004. Zlín: Univerzita Tomáše Bati ve Zlíně, 1. vyd. 250 s. ISBN 80-7318-189-4.
- [OBJ11] OBJECT MANAGEMENT GROUP: Business Process Model and Notation. 2011. In: *omg.org* [online]. [Cit. 13. 3. 2020]. Dostupné na internete: <<http://www.omg.org/spec/BPMN/2.0/PDF/>>.
- [ROB98] ROBSON, M. – ULLAH, P.: *Praktická príručka podnikového reengineeringu*. 1. vyd. Praha: Management Press, 1998. 178 s. Z angl. orig. preložil Pavel Medek. ISBN 80-85943-64-6.
- [SCH10] SCHMIDT, P.: *Procesný prístup k IS organizácie*. 2010. Dizertačná práca. Bratislava: FHI EUBA, 134 s.

[STA10] STAŠÁK, J.: *Modelovanie systému riadenia ekonomických objektov*. 2010.  
Bratislava: Vydavateľstvo Ekonóm. 182 s. ISBN 978-80-225-2896-2.

[TRU99] TRUNEČEK, J.: *Management v informační společnosti : učební texty pro bakalářské studium*. 1999. Praha: Vysoká škola ekonomická v Praze. 2.vyd. 228 s. ISBN 8070796839.

## Kapitola 2

### Servisne orientovaná architektúra a jej základné charakteristiky

V kapitole 1 sme detailne rozobrali problematiku funkčného a procesného riadenia podniku, ktorej pochopenie predstavuje základné východisko k správnemu návrhu podnikového informačného systému. V kapitole 2 sa zase zameriame na servisne orientovanú architektúru ako na jeden zo spôsobov, za pomoci ktorých sa dá takýto systém vybudovať.

#### 2.1 Východiskové pojmy a fakty

Táto podkapitola predstavuje stručný úvod do problematiky informačných systémov a servisne orientovanej architektúry.

##### 2.1.1 Servisne orientovaná architektúra

**Servisne orientovaná architektúra (SOA)** je myšlienkový koncept, resp. prístup k vytváraniu a integrácii informačných systémov, ktorý je založený na používaní webových služieb a na ich vzájomnej komunikácii štandardizovaným spôsobom.

Podstatou je snaha o rozdelenie väčšieho a zložitejšieho problému, resp. systému na viacero menších a ľahšic zvládnuteľných podproblémov, resp. podsystémov. Ide o dobre známy princíp *dekompozícia* zložitejšieho celku na niekoľko menších a jednoduchších častí, ktoré sú navzájom previazané. V tomto prípade sa informačný systém dekomponuje na množinu služieb schopných vzájomnej spolupráce. *Pojem služba treba v súvislosti so SOA chápať ako aplikáciu, t. j. počítačový program.* O riešenie jedného podproblému sa nemusí staráť iba jedna služba, ale toto riešenie môže byť zabezpečované spoluprácou viacerých služieb, z ktorých každá sa sústredí na určitú časť podproblému. Systém sa teda dekomponuje na podproblémy, ktoré sa dajú podľa potreby ďalej dekomponovať. Je to jeden zo základných princípov štruktúrovaného vývoja softvéru, a preto v oblasti softvérového inžinierstva nejde o žiadnu novinku. SOA je však okrem tohto základného princípu utváraná aj množstvom ďalších charakteristických znakov, čo ju odlišuje od tradičných prístupov k tvorbe softvéru.

##### 2.1.2 Údaje, informácie, znalosti

###### Údaje

Spomedzi viacerých definícií pojmu *údaj* vyberá Závodný vo svojej publikácii *Riadenie projektov informačných systémov* nasledujúcu: „*Údaj je vyjadrenie skutočnosti a myšlienok*

*v predpísanej podobe tak, aby ich bolo možné prenášať a spracovať.“* [ZÁV06]. Kokles a Romanová v publikácii *Informačný vek* dodávajú: „*Dáta (údaje) sú tvorené číselnými, abecednými alebo abecednočíselnými znakmi. Väčšinou majú vopred určený vnútorný zmysel, rozsah (minimálny a maximálny počet miest) a označenie. Izolovaný údaj bez definovania jeho charakteristiky alebo bez spojenia s ďalšími údajmi obvyčajne nemožno prakticky použiť.*“ [KOK02]. Lexikón informatiky k tomu ešte dodáva, že v informatike by údaje mali byť reprezentované alebo zapísané takým spôsobom, aby sa dali strojovo rozpoznávať a automatizované spracovať [CLA91]. Pre úplnosť dodávame, že v množnom číslе je vhodnejšie v slovenčine používať pojem *údaje*, ktorý je z jazykového hľadiska správnejší, než pojem *dáta*.

## **Informácie**

S pojmom *údaj* úzko súvisí pojem *informácia*. Informácie sa často definujú ako výsledky spracovania údajov prezentované v takej podobe, aby ich príjemca, pre ktorého sú určené, mohol nejakým spôsobom použiť. Existuje však mnoho rozličných definícií pojmu informácia, pričom ich vhodnú sumarizáciu poskytuje Závodný vo svojej publikácii *Riadenie projektov informačných systémov*, kde uvádza: „*Existujú tri názorové príudy na definíciu informácie:*

1. *štatistické ponímanie informácie, vychádzajúce z teórie informácie, podávajúce presný a merateľný výklad pojmu informácia ako miery zníženia entropie (t. j. nevedomostí) pri rozhodovaní;*
2. *pragmatický smer, ktorý považuje pojmy údaj, informácia a správa za synonymá;*
3. *smer ponímania informácie, podľa ktorého až rozhodovací proces zhodnocuje údaje na informácie.*“ [ZÁV06].

Tretí z týchto troch smerov hovorí, že údaj predstavuje pre príjemcu informáciu vtedy, ak pre neho prináša niečo nové (čo príjemca doteraz nevedel) a zároveň zaujímavé (príjemca má záujem o tento údaj). Z tohto pohľadu je teda každá informácia súčasne údajom, ale každý údaj nemusí byť u konkrétneho príjemcu zhodnotený na informáciu.

## **Znalosti a vedomosti**

Znalosti a vedomosti predstavujú najvyššiu formu abstrakcie poznania. Závodný tieto pojmy definuje nasledovným spôsobom [ZÁV06]:

- „*Znalosť*“ možno charakterizovať ako súhrn teoretických poznatkov, predstáv, pojmov a teórií nadobudnutých výskumom, učením, praktickou činnosťou a skúsenosťami z danej oblasti ľudskej činnosti.
- *Vedomosť*, pojem často používaný ako synonymum k terminu *znalosť*, je výsledkom poznávacieho procesu. Je to teda výsledok poznávania, vnímania, myšlenia, praktického experimentovania, riešenia problémov a zdolávania prekážok. Vyjadruje pochopenie a porozumenie informáciám tak, že z nich umožňuje odvodzovať závery“.

So znalosťami úzko súvisia pojmy *znalostný* a *expertný* systém. **Znalostný systém** je systém, ktorý pracuje na princípoch umelej inteligencie a často sa označuje aj ako inteligenčný systém. Je schopný získať znalosti, reprezentovať ich, ukladať do bázy znalostí (Knowledge Base) a následne ich prostredníctvom inferenčného mechanizmu (Inference Engine) distribuovať oprávneným používateľom, generovať nové znalosti na základe existujúcich a sám sa učiť.

Ak je *znalostný* systém úzko predmetne orientovaný, nazýva sa **expertný systém**. Takýto systém je schopný nahradzať, resp. zastúpiť expertov z určitej konkrétnnej tematickej oblasti (napr. medicína, správa jadrového reaktora, matematika, geológia, chémia a pod.). Expertné systémy sú použiteľné aj v podnikovej sfére, a to najmä na podporu riešenia neštruktúrovaných (nedostatočne formulovaných, nejasných) problémov, pri riešení ktorých sa vyžadujú určité zručnosti, skúsenosti a „manažérské cítenie“, ako napr.:

- stanovovanie vhodnej podnikovej stratégie,
- rozhodovanie o vstupe určitých produktov na trh,
- výber najvhodnejšieho zamestnanca na prijatie či ukončenie pracovného pomeru,
- vyhodnocovanie „zdravotného stavu“ podniku, t. j. či sa podnik nachádza v útlme, v stagnácii alebo, naopak, v rozkvete, či neexistujú určité indikátory toho, že mu hrozí bankrot a pod.,
- tvorba investičného plánu, ktorý je podniku „šitý na mieru“ a pod.

Problematika *znalostných* a *expertných* systémov presahuje rámec tejto publikácie, a preto sa ňou nebudeme ďalej zaoberať. Záujemcom o hlbšie preniknutie do tejto oblasti odporúčame napr. literárny zdroj [NÁV07], pričom základné informácie sa dajú nájsť aj v [JUR18].

### *2.1.3 Informačný systém a jeho architektúra*

Pojem **systém** vo všeobecnosti označuje množinu navzájom súvisiacich prvkov a množinu väzieb, resp. vzťahov medzi týmito prvkami. S tým úzko súvisí pojem **architektúra systému**, ktorý vyjadruje, z akých prvkov sa určitý systém skladá a aké sú väzby, resp. vzťahy medzi týmito prvkami. V zmysle teórie systémov môžeme za systém preto považovať napr. výrobný podnik, školu, mestský úrad, banku, informačný systém, športový oddiel so svojimi členmi a funkcionármi a pod.

Na podnik sa dá nahliaľať ako na systém z rôznych uhlov pohľadu. Z organizačného hľadiska ide o množinu vzájomne previazaných organizačných útvarov. Z funkcionálneho hľadiska ide o množinu vzájomne súvisiacich funkcií. A z procesného hľadiska ide zasa o množinu súvisiacich procesov. Pri každom z týchto troch uhlov pohľadu je systém (v tomto prípade podnik) tvorený množinou určitých komponentov, ktoré sú vzájomne previazané väzbami vyjadrujúcimi ich vzájomnú nadradenosť, podradenosť alebo rovnocennosť.

**Informačný systém** môžeme definovať ako *systém na zber, uchovávanie, spracovanie, prenos, analýzu a poskytovanie údajov, resp. informácií*. Podnikový informačný systém sa vo všeobecnosti považuje za nástroj informačnej podpory podnikových procesov. Súčasťou informačného systému sú najmä:

- ľudia – používatelia, tvorcovia, administrátori,
- hardvér,
- softvér,
- telekomunikácie,
- databázy,
- papierové dokumenty ako nosiče údajov,
- procesy (presné postupy vykonávania určitých činností),
- inštrukcie a požiadavky,
- zdroje údajov,
- algoritmy a metódy.

Pri podnikovom informačnom systéme rozlišujeme jeho *globálnu architektúru a niekoľko čiastkových architektúr*. **Globálna architektúra** vychádza z toho, že v podnikoch sa zvyknú vyskytovať tri úrovne riadenia – strategická, taktická a operatívna úroveň, pričom informačný systém musí poskytovať podporu manažérom na všetkých týchto úrovniach.

*Strategická úroveň* je najvyššia a je tvorená vrcholovým manažmentom firmy, ktorý stanovuje jej základnú orientáciu, predmetné oblasti podnikania a hlavné ciele. Vrcholový manažment podniku typicky tvoria [MAN11]:

- *riaditeľ spoločnosti* – vo veľkých spoločnostiach sa zvykne označovať ako generálny riaditeľ alebo tiež CEO (Chief Executive Officer),
- *finančný riaditeľ* (Chief Financial Officer; CFO),
- *prevádzkový riaditeľ* (Chief Operations Officer; COO),
- *IKT riaditeľ* (riaditeľ pre informačné a komunikačné technológie; Chief Information Officer; CIO),
- *personálny riaditeľ* (Chief Human Resources Officer; CHRO),
- *obchodný riaditeľ* (Chief Sales Officer; CSO),
- podľa organizačnej štruktúry a potrieb podniku sem môžu patriť aj iní manažéri – napr. *technický riaditeľ*, *výrobný riaditeľ*, *riaditeľ závodu*, *riaditeľ pobočky*, *riaditeľ regiónu* a pod.

Manažéri pracujú na tejto úrovni s agregovanými údajmi (napr. súčty, priemery či špeciálne ukazovatele vypočítavané na základe vzorcov), ktoré môžu ísť aj hlboko do minulosti (t. j. s aktuálnymi, ale aj historickými údajmi). Pri plánovaní sa zameriavajú na dlhodobý časový horizont (rok alebo niekoľko rokov dopredu). Pri stanovovaní predpovedí vývoja situácie na trhoch a stanovovaní cieľov a stratégii podniku sa môžu opierať o expertné systémy, ktoré sú svojou povahou vhodné na riešenie ľažko algoritmizovateľných úloh. *Časť informačného systému, ktorá slúži na podporu činnosti vrcholového manažmentu, sa označuje ako EIS (Executive Information System).*

*Taktická úroveň riadenia* je tvorená stredným manažmentom, t. j. riaditeľmi, ktorí sa nachádzajú v stredných častiach organizačnej štruktúry. Ako uvádza server ManagementMania.com: „*rozsah ich zodpovednosti je na úrovni väčších organizačných jednotiek alebo určitej oblasti, ktorá ide naprieč organizáciou. Pre menšie organizácie platí pravidlo, že vrcholový manažment je tvorený iba riaditeľom spoločnosti a všetky ďalšie vyššie vymenované pozicie riaditeľov sú na úrovni stredného manažmentu.*“ [MAN11]. Vo väčších organizáciách do stredného manažmentu typicky patria napr. [MAN11]:

- *riaditeľ kvality,*
- *riaditeľ rizík,*

- riaditeľ bezpečnosti,
- riaditeľ vývoja a mnohí iní.

Stredný manažment prijíma pokyny, ciele a základné stratégie stanovené vrcholovým manažmentom a musí sa nimi riadiť. Pri plánovaní sa zameriava na strednodobý časový horizont (mesiac alebo niekoľko mesiacov dopredu). Pri rozhodovaní pracuje s menej agregovanými údajmi (v porovnaní s potrebami vrcholového manažmentu), ktoré nezachádzajú tak d'aleko do minulosti. Časť informačného systému, ktorá slúži na podporu činnosti stredného manažmentu, sa označuje ako MIS (Management Information System).

*Operativna úroveň riadenia* je tvorená najnižším manažmentom, ktorý sa tiež označuje ako operatívny manažment alebo aj manažment prvej línie. Ide o manažérov, ktorí sú v organizačnej štruktúre najnižšie – zvyčajne na koncoch jej vetiev. Medzi takýchto manažérov typicky patria napr. [MAN11]:

- produktový manažér,
- manažér logistiky,
- manažér marketingu,
- manažér služieb,
- facility manažér (je zodpovedný za vytváranie priaznivého pracovného prostredia naprieč celou organizáciou),
- správca budov,
- vedúci učtárne,
- vedúci skladu a iní.

Operatívny manažment je v organizačnej štruktúre najnižšie a musí sa riadiť pokynmi a požiadavkami z vyšších úrovní riadenia. Pri plánovaní sa zameriava na krátkodobý časový horizont (deň alebo niekoľko dní, často ide o plánovanie na jeden týždeň). Pri rozhodovaní pracuje s údajmi, ktoré sú menej detailné (t. j. nie sú agregované, resp. summarizované, pretože také údaje nemajú pre operatívny manažment veľký význam – sú potrebné konkrétné údaje za konkrétné dni) a aktuálne (z hľadiska operatívneho plánovania nemá zmysel zachádzať do histórie a treba sa riadiť aktuálnou situáciou). Časť informačného systému, ktorá slúži na podporu činnosti operatívneho manažmentu, sa označuje ako TPS (Transaction Processing System). Viac o TPS sa dá dočítať v [JUR18].

Ako sme už spomenuli, okrem globálnej architektúry má informačný systém aj viacero čiastkových architektúr, ktoré vyjadrujú jeho štruktúru z určitých špecifických uhlov pohľadu. Medzi čiastkové architektúry môžeme zaradiť najmä tieto:

- **funkčná architektúra IS** – pod pojmom funkcia IS by sme mali rozumieť určitú činnosť, ktorú IS na rutinnej báze vykonáva, resp. zabezpečuje a ktorá slúži na podporu činnosti konkrétnego organizačného útvaru alebo konkrétnej osoby v podniku. V kapitole 1.2 sme spomínali hierarchický diagram funkcií, ktorý predstavuje hierarchicky usporiadaný zoznam všetkých funkcií jednotlivých organizačných útvarov v podniku. Pri dodržaní rovnakých princípov a rovnakej notácie vieme zostaviť *hierarchický diagram informačných funkcií*, v ktorom sú zakreslené iba funkcie IS. Tento diagram potom reprezentuje funkčnú architektúru IS.
- **Procesná architektúra IS** – z hľadiska procesného prístupu k riadeniu podniku slúži IS na podporu vykonávania jednotlivých podnikových procesov, na ich skvalitnenie, zníženie ich chybovosti a pod. Na modelovanie horizontálnej a vertikálnej štruktúry podnikových procesov sa dajú použiť diagramové techniky opísané v kapitole 1 (t. j. napr. *vývojové diagramy*, *BPMN diagramy*, *procesné mapy*, *Juríkove-Schmidtove diagramy* a iné). V tejto súvislosti je potrebné zdôrazniť, že IS nemusí automatizovať, resp. podporovať vykonávanie každého čiastkového kroku v každom procese. Mnohé kroky môžu byť vykonávané manuálne zodpovednými pracovníkmi bez použitia IS.
- **Dátová architektúra IS** – základom každého informačného systému je databáza a dátová architektúra vyjadruje, aké dátové množiny sú v databáze uchovávané a aké sú medzi nimi vzťahy. Zjednodušene povedané: dátová architektúra vyjadruje, o čom uchovávame údaje, akým spôsobom ich uchovávame a ako sú tieto údaje previazané. Vhodnými prostriedkami na znázornenie dátovej architektúry IS je *entitno-relačný diagram* (Entity Relationship Diagram; ERD) a *diagram dátových tokov* (Data Flow Diagram; DFD). Dátová architektúra sa však dá reprezentovať aj *koncepcuálnym, logickým a fyzickým dátovým modelom*. Viac o týchto modeloch sa dá dočítať v [JUR18].
- **Aplikačná architektúra IS** – informačný systém býva zvyčajne tvorený prepojením viacerých čiastkových aplikácií do jedného spolupracujúceho celku. Aplikačná architektúra IS vyjadruje, z akých aplikácií IS pozostáva a ako sú spolu prepojené. Zvyčajne je totiž zbytočné prepájať každú aplikáciu s každou (tzv. špagetová

integrácia), pretože by to viedlo k zbytočne komplikovanému systému, ktorý by bol náročný na údržbu. Aplikácie treba vzájomne prepájať iba vtedy, ak má toto spojenie praktický zmysel. Aplikácie v IS pritom môžu byť rôzneho pôvodu – môže byť medzi nimi *typový aplikáčny softvér* (TASW), ktorý je určený pre všeobecného zákazníka a je zostavený na základe osvedčených postupov (best practices) v určitom odvetví alebo oblasti; *individuálny aplikáčny softvér* (IASW), ktorý je vytvorený pre konkrétnu firmu na základe jej individuálnych požiadaviek alebo *open-source softvér* (OSS), ktorý býva vyvíjaný komunitami softvérových nadšencov pre ľubovoľných používateľov za dobrovoľný alebo žiadny poplatok, no máva zvyčajne obmedzenia vo funkcionalite a nebýva taký otestovaný a dôveryhodný ako TASW a IASW.

- **Softvérové architektúry jednotlivých aplikácií v IS** – každá aplikácia, ktorá je súčasťou IS, má vlastnú vnútornú štruktúru, ktorú nazývame jej softvérovou architektúrou. Aplikácie sa zvyknú skladať z troch základných častí: *dátová časť* (databáza), *aplikáčna časť* (naprogramované funkcie, ktoré operujú nad databázou a vykonávajú rozličné úkony podľa požiadaviek používateľa) a *používateľské rozhranie* (umožňuje používateľovi pracovať s aplikáciou, t. j. zadávať vstupné údaje a inštrukcie a prijímať výstupy). Podľa spôsobu realizácie na klientskom zariadení a serveri rozlišujeme viaceré riešenia softvérovej architektúry: *monolitická architektúra* (dátová časť, aplikačná časť aj používateľské rozhranie sa realizujú na tom istom zariadení), *dvojvrstvová architektúra klient/server typu tenký klient* (na klientskom zariadení sa prevádzkuje iba používateľské rozhranie – zvyčajne postačuje webový prehliadač; zatiaľ čo funkcie aplikačnej vrstvy sa vykonávajú na serveri, na ktorom sa prevádzkuje databáza), *dvojvrstvová architektúra klient/server typu tučný klient* (na serveri sa prevádzkuje iba databáza a funkcie aplikačnej vrstvy sa vykonávajú na klientskom zariadení, na ktorom sa prevádzkuje aj používateľské rozhranie – zvyčajne nepostačuje webový prehliadač a na klientskom zariadení musí byť nainštalovaná špeciálna aplikácia), *trojvrstvová architektúra klient/server* (databáza sa prevádzkuje na databázovom serveri, funkcie aplikačnej vrstvy sa vykonávajú na aplikačnom serveri a na klientskom zariadení sa prevádzkuje iba používateľské rozhranie – zvyčajne postačuje webový prehliadač) či *viacvrstvové architektúry*. Aplikácia sa tiež môže vnútornie skladáť z menších aplikačných častí – modulov, ktoré navzájom spolupracujú a dohromady vytvárajú jeden celok – funkčnú aplikáciu. Podľa spôsobu prepojenia modulov a ich nadväznosti rozlišujeme nasledovné *štiri základné modely usporiadania*

*modulov v aplikácii: lineárny model, hierarchický model, vrstvový model a sieťový model.* Viac o týchto modeloch sa dá dočítať v [JUR18].

- **Hardvérová architektúra IS** – ide o množinu všetkých technických a komunikačných prostriedkov, ktoré sú súčasťou IS. Patria sem najmä servery, koncové stanice (t. j. počítače, resp. zariadenia, na ktorých pracujú koncoví používatelia systému) a prvky súvisiace so sieťovou infraštruktúrou – modemy, routre, switche, rackové skrine, kabeláž atď. Vhodným nástrojom na reprezentáciu hardvérovej architektúry je *diagram rozmiestnenia* (deployment diagram).

V dnešnej dobe si pod pojmom informačný systém takmer každý predstaví niečo elektronické, ale nie vždy to bolo tak. O existencii informačných systémov môžeme totiž hovoriť už v časoch, keď ľudstvo ešte vôbec nepoznalo počítače. Hlavným pamäťovým médiom na uchovávanie informácií v podnikovej sfére pred zavedením počítačov bol papier a za ekvivalent databáz môžeme považovať najmä kartotéky a archívy rozličných dokumentov. K spracovaniu údajov a ich transformácií na informácie vtedy dochádzalo najmä prostredníctvom manuálnych výpočtov a analýz, ktoré vykonávali ľudia, a nie počítače. V podnikoch teda už aj v časoch pred vynájdením počítačov existoval určitý *zaužívaný spôsob, resp. systém zbierania, uchovávania a spracovávania údajov*. Zavedením počítačov došlo k elektronizácii tohto systému. Elektronický informačný systém umožňuje automatizáciu mnohých činností súvisiacich so zbieraním, uchovávaním, spracovávaním a poskytovaním údajov. Tým dochádza k určitému odbremenneniu ľudskej pracovnej sily. Zavedením elektronických informačných systémov dochádza k zbieraniu väčšieho množstva údajov ako v minulosti, čo otvorilo dvere realizácií zložitejších analýz a výpočtov, ktoré by ľudia v mnohých prípadoch pri manuálnej realizácii nemohli v reálnom čase uskutočniť [JUR18].

V súvislosti s hromadením obrovských kvánt údajov sa v ostatných rokoch často používa pojem „big data“ (t. j. „veľké údaje“). Spoločnosť Gartner, Inc. definuje „big data“ ako „*velkoobjemové, vysokorychlosné a/alebo veľmi rozmanité (mnohotvárne) informačné aktíva, ktoré si vyžadujú nákladovo efektívne a inovatívne formy ich spracovania, aby nám umožnili získanie lepšieho náhľadu na určitý problém, pomohli pri rozhodovaní a automatizácii procesov.*“ [GAR18]. Podľa Infostatu existuje viacero rozličných definícií pojmu „big data“, ale takmer vo všetkých sa spomínajú tri základné charakteristiky [SUJ14]:

- *Množstvo (Volume)* – typické sú veľké objemy údajov, ktoré spravidla začínajú na desiatkach terabajtov.

- *Rýchlosť (Velocity)* – rýchlosť, s akou údaje vznikajú, príp. rýchlosť, s akou sú spracovávané.
- *Pestrosť (Variety)* – rôznorodosť údajových formátov vo vnútri „big data“.

## 2.2 SOA – základné princípy a charakteristiky

Ako uvádzá Erl vo svojej publikácii: „*Servisne orientovaná architektúra je termín, predstavujúci model, v ktorom je logická automatizácia rozdelená na menšie odlišné jednotky logiky. Dohromady tieto jednotky tvoria väčší kus logiky automatizácie riadenia.*“ [ERL09]. *Zjednodušene povedané, funkciionalita informačného systému ako súhrn všetkého, čo systém dokáže vykonávať alebo zabezpečovať, by sa mala rozdeliť na množstvo malých logických jednotiek, ktoré sú autonómne, ale navzájom spolupracujú. Tieto logické jednotky sú reprezentované webovými službami.* Činnosti, ktoré sú zabezpcované jednotlivými službami, sa môžu odlišovať z hľadiska svojho rozsahu a zložitosti. Okrem toho môže logika určitej služby zahŕňať aj logiku inej služby. To znamená, že takáto služba nemôže fungovať úplne samostatne a pre zabezpečovanie činností, pre ktoré je určená, musí spolupracovať s nejakou inou službou, ktorá jej poskytne určité medzivýsledky, alebo zabezpečí vykonanie istých čiastkových úkonov.

Služby sa teda môžu navzájom výrazne odlišovať z hľadiska rozsahu činností, ktoré vykonávajú, t. j. z hľadiska rozsahu logiky, ktorú reprezentujú. Pod pojmom **logika služby** rozumieme pomenovanie toho, čo má služba zabezpečovať (t. j. čo je jej poslaním) vrátane spôsobu, ako to má robiť (teda aké čiastkové úkony musí vykonávať na to, aby dosahovala požadované výsledky, resp. napĺňala svoje poslanie). *V súvislosti s procesným riadením podnikov, resp. organizácií môže služba reprezentovať:*

- IT podporu určitého podnikového procesu ako celku,
- IT podporu určitého podprocesu v rámci nejakého širšieho podnikového procesu,
- IT podporu jedného konkrétneho kroku alebo skupiny krovov v rámci podnikového procesu alebo podprocesu.

V SOA teda jedna služba môže používať inú službu. Podmienkou takejto spolupráce je však to, že služby musia byť navzájom informované o svojej existencii. Na tento účel slúžia tzv. *opisy služieb*. **Opis služby** stanovuje názov a umiestnenie určitej konkrétnej služby, ako aj

spôsob, akým je možné s touto službou komunikovať. Platí, že služba A si je vedomá existencie služby B vtedy, ak služba A má prístup k opisu služby B.

Na to, aby služby mohli navzájom spolupracovať, musí medzi nimi prebiehať výmena informácií. K tomu dochádza na základe *mechanizmu výmeny správ*. Správy sú nezávislými jednotkami komunikácie, ktoré „sú schopné sa o seba postarať“. To znamená, že len čo jedna služba odošle druhej službe určitú správu, stráca nad touto správou kontrolu a nemôže už ovplyvniť jej ďalšie smerovanie.

Existuje **8 základných princípov SOA**, medzi ktoré patria (spracované podľa [SOA17] a [ERL09]):

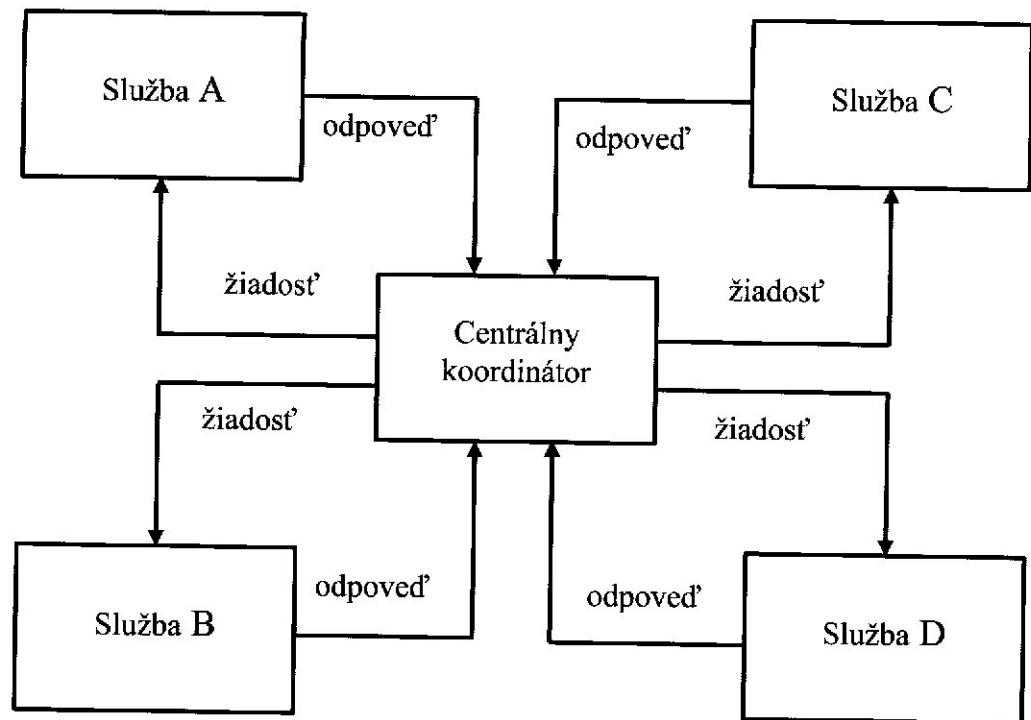
1. *Princíp dekompozície zložitejšieho problému na jednoduchšie a ľahšie zvládnuteľné podproblémy* – tento princíp sme vysvetlili na začiatku tejto kapitoly.
2. *Najskôr sa vytvárajú dokumenty opisujúce službu, až potom sa programuje jej funkcia* – skôr než sa naprogramuje logika určitej služby a skôr než sa táto služba uvedie do praxe, je potrebné vytvoriť tzv. opis služby. V terminológii SOA sa tento opis označuje ako kontrakt (slovo „contract“ môžeme z angličtiny preložiť ako „zmluva“ alebo „dohoda“). Tento kontrakt môže byť tvorený viacerými dokumentmi, predovšetkým WSDL dokumentom, XML schémami zapísanými v jazyku XSD, prípadne inými dokumentmi opisujúcimi správanie sa tej-ktorej služby a spôsob jej komunikácie s okolím (napr. SLA – Service Level Agreement; ide o dohodu uzavorenú medzi poskytovateľom služby a jej používateľom týkajúcu sa kvality služby, podmienok jej používania, dostupnosti, zodpovednosti za prípadné škody a pod.). Podstatou tohto princípu je teda to, že najskôr sa vytvorí kontrakt služby ako séria dokumentov, ktoré ju opisujú, a až potom sa vytvára (programuje) vlastná logika tejto služby.
3. *Zistiteľnosť služieb (snaha o to, aby služba bola viditeľná a dostupná pre iné služby, resp. pre okolity svet)* – základnou podmienkou toho, aby mohla byť určitá služba opäťovne používaná inými službami je, že musí byť viditeľná pre iné služby, t. j. iné služby musia mať k dispozícii informácie o jej existencii a o tom, ako sa s touto službou spojiť. Základným dokumentom opisujúcim určitú službu je jej WSDL opis, no dôležité môžu byť aj iné dokumenty (XML schémy zapísané v jazyku XSD, SLA dokumenty a pod.). Tieto dokumenty by mali byť uložené v úložisku, ku ktorému majú ostatné služby prístup a môžu v ňom vyhľadávať iné služby podľa špecifických kritérií.

Takýmto úložiskom môže byť predovšetkým register služieb UDDI, ktorý budeme bližšie charakterizovať v kapitole 6.4. V záujme naplnenia princípu zistiteľnosti služieb by teda mali všetky služby sprístupniť svoje opisy tak, aby sa o ich existencii mohli dozvedieť iné služby alebo používatelia a aby ich mohli bez problémov spustiť.

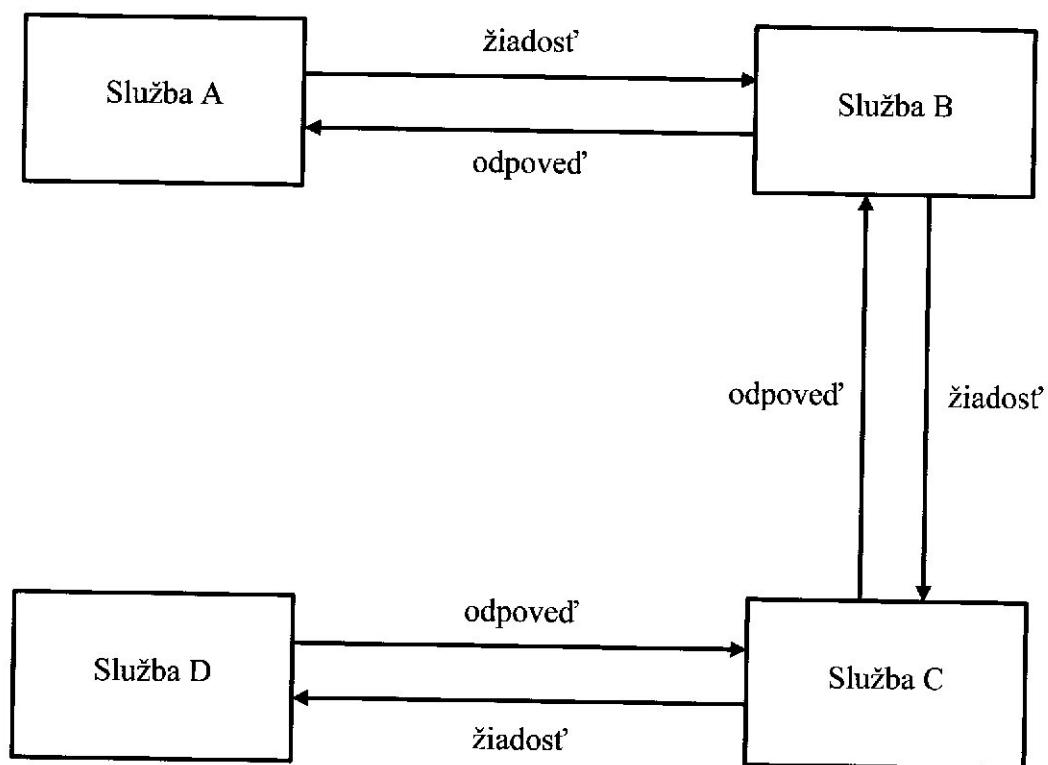
4. *Abstrakcia* – služba by navonok mala pôsobiť ako „čierna skrinka“, pri ktorej nie je okolitému svetu celkom jasné, čo sa presne deje vo vnútri. Je len jasné, aké vstupy táto služba prijíma, aké výstupy poskytuje a ako je možné s ňou komunikovať. Konkrétnie implementačné detaile však zostávajú skryté. Vďaka tomu napr. nemusí byť ostatným službám, ktoré sa na určitú službu obracajú, zrejmé, či sa táto služba v skutočnosti obracia na iné služby so žiadostou o vykonanie čiastkových operácií, a teda či je v plnej miere sebestačná alebo nie. Kontrakt služby by teda nemal prezrádzať, ako presne služba zabezpečuje transformáciu vstupov na výstupy, aké kroky a v akom poradí pritom vykonáva a či je v plnej miere sebestačná. Princíp abstrakcie sa niekedy označuje aj ako *princíp voľnej väzby medzi kontraktom služby a logikou služby*.
5. *Snaha o znovupoužiteľnosť služieb* – táto charakteristika sa nevzťahuje na to, ako často sa určitá služba používa, keďže všetky služby sú navrhované tak, aby mohli byť používané opakovane a nielen jednorazovo, ale vzťahuje sa na to, do akej miery môže byť táto služba použitá na automatizáciu viacerých podnikových procesov. Ak sa dá služba použiť pri automatizácii len jedného procesu, potom je jej znovupoužiteľnosť nulová. Naopak, čím vyšší je počet procesov, pri ktorých automatizáciu sa táto služba dá využiť, tým vyššia je aj jej znovupoužiteľnosť. Znovupoužiteľnosť služby je úzko spätá s tým, do akého modelu patrí táto služba (poznámka: v kapitole 3.3 sa oboznámíme s tým, že existujú viaceré modely služieb, medzi ktoré patria služby entitné, služby pomocné a služby úzko späté s podnikovými procesmi, ktoré sa členia na koordinačné a spracovateľské). Najmä pri službách úzko spätých s konkrétnym podnikovým procesom je neraz ľažké dosiahnuť ich znovupoužiteľnosť. Jedným zo spôsobov, ako to urobiť, je snaha o to, aby služba nedisponovala len tou funkcionálitou, ktorá sa zdá byť nevyhnutná pri jej vytváraní, ale aby ponúkala aj určitú funkcionálitu navyše. Čím viac rôznych funkcií určitá služba ponúka, tým vyššia je šanca, že ju budú môcť využiť viaceré procesy. Takéto riešenie má však aj svoju nevýhodu, ktorou je veľké množstvo kódov s otáznou využiteľnosťou (čo zbytočne predlžuje čas potrebný na dokončenie služby a tiež náklady spojené s jej tvorbou), a preto je jeho vhodnosť sporná.

6. *Služby sú schopné vzájomnej spolupráce, aby dohromady vytvárali zmysluplný celok – ak nemáme k dispozícii službu, ktorá by sama o sebe bola schopná zabezpečiť dosiahnutie požadovaného výsledku, môžeme sa pokúsiť dosiahnuť tento výsledok vzájomnou spoluprácou dvoch alebo viacerých služieb. Proces logického zlučovania viacerých služieb do jedného zmysluplného celku sa označuje ako **skladanie služieb**, resp. **kompozícia služieb**. Pri kompozícii nedochádza k fyzickému zlučovaniu služieb, ale len k logickému zlučovaniu (čiže k spolupráci samostatných programov).* To znamená, že každá služba si ponecháva svoju nezávislosť a jedinečnosť a existuje aj nadálej ako samostatne spustiteľná služba. Kompozícia sa teda vzťahuje na situácie, pri ktorých jedna webová služba zabezpečuje podporu jednej alebo viacerým službám na to, aby mohli úspešne dokončiť svoju úlohu, alebo koordinuje ich spúšťanie v správnom poradí, a tiež o situácie, keď sa jedna služba obracia na inú službu, aby zabezpečila vykonanie nejakej čiastkovej úlohy. Každá služba, ktorá sa zúčastňuje na takejto kompozícii, prijíma úlohu *člena kompozicie*. Pri návrhu nových webových služieb je veľmi dôležité zamyslieť sa nad kompozíciami, ktorých má byť táto služba súčasťou, aby sa dosiahla čo najvyššia efektivita spolupráce medzi službami.

Poznáme *dva základné typy kompozicie služieb: choreografiu a orchestráciu*. Pri **orchestrácii** služby odovzdávajú svoje výstupy centrálnemu koordinátorovi (t. j. centrálnej riadiacej službe), ktorý rozhoduje o tom, ktorá služba bude ďalej spustená, a následne jej odovzdáva potrebné vstupy. Centrálna riadiaca služba zabezpečuje, aby sa všetky služby kompozicie spustili v správnom poradí tak, aby bol dosiahnutý požadovaný výsledok, resp. aby bol dodržaný správny tok určitého podnikového procesu alebo jeho časti (podprocesu). **Choreografia** je zasa taký typ kompozicie služieb, pri ktorom sa nevyužíva centrálny koordinátor a každá zúčastnená služba je aj bez koordinátora schopná presne rozpoznať, kedy sa má spustiť, a s akými inými službami má komunikovať. Pravidlá, ktoré hovoria, ako sa majú spúšťať na seba nadväzujúce služby, pri choreografii teda nie sú prenesené na centrálnu riadiacu službu, ale sú zakomponované priamo do zdrojového kódu jednotlivých služieb. Ku komunikácii medzi zúčastnenými službami dochádza pri choreografii a aj pri orchestrácii na základe mechanizmu zasielania správ, ktoré sú vytvorené v štandardizovanom formáte a sú zasielané štandardizovaným spôsobom. Tento mechanizmus je podrobnejšie opísaný v ďalších kapitolách. Rozdiel medzi orchestráciou a choreografiou služieb znázorňujú obr. 15 a obr. 16.



Obr. 15: **Orchestrácia služieb** [Zdroj: autor]



Obr. 16: **Choreografia služieb** [Zdroj: autor]

7. *Snaha minimalizovať závislosť jednotlivých služieb od externých zdrojov (napr. databáz, výkonu procesora, operačnej pamäte a pod.) a dosiahnuť tak ich vysokú autonómiu* – tento princíp úzko súvisí s princípom znovupoužiteľnosti služieb, podľa ktorého by sa služby mali podieľať na automatizácii čo najväčšieho počtu podnikových procesov. Pri veľkej frekvencii používania sú však kladené veľké nároky na *spoločnosť služby*. Tá závisí od schopnosti služby mať pod kontrolou vlastnú logiku, ktorú táto služba zapuzdruje, a tiež ďalšie zdroje tak, aby bola len minimálne závislá od externých zdrojov, ktoré nemá pod kontrolou, alebo ich má pod kontrolou len čiastočne. Takými zdrojmi sú napr. spoločná databáza, o ktorú sa delia viaceré služby, alebo spoločné spracovateľské kapacity (procesorový výkon, disponibilná operačná pamäť), ktoré nemusia byť k dispozícii práve vtedy, keď ich táto služba potrebuje. Napr. ak určitá služba vykonáva úlohy náročné na operačnú pamäť alebo výkon procesora, potom by mala byť táto služba prevádzkovaná na samostatnom serveri s dostatočnými výpočtovými kapacitami, príp. by mala mať výpočtové kapacity do určitej miery rezervované. Závislosť od centrálnej databázy, ktorá je spoločne využívaná viacerými službami, sa dá zredukovať poskytovaním lokálnych kópií potrebných údajov jednotlivým službám tak, aby ich mali stále k dispozícii. Pritom je však veľmi dôležité zabrániť stavu, pri ktorom by jednotlivé služby disponovali navzájom neidentickými údajmi, z ktorých niektoré by boli nesprávne (t. j. je potrebné zabezpečiť vzájomnú synchronizáciu všetkých lokálnych kópií údajov z centrálnej databázy).
8. *SOA je založená na otvorených štandardoch* – výmena údajov medzi službami v rámci SOA sa riadi otvorenými štandardmi, a to jednak štandardmi určujúcimi štruktúru prenášaných správ (XML, XSD, SOAP) ale aj štandardmi, prostredníctvom ktorých sa riadi ich prenos (http, MIME) či štandardmi používanými na opis služieb (WSDL). Po odoslaní určitej správy konkrétnou službou putuje táto správa ďalej samostatne podľa pravidiel prenosového protokolu, ktorý sa na tento prenos používa. *Otvorený štandard* je publikovaná (t. j. zverejnená) špecifikácia na uchovávanie alebo prenos digitálnych údajov spravidla vytvorená štandardizačnou organizáciou, ktorá netrvá na žiadnych právnych obmedzeniach pri používaní tejto špecifikácie inými firmami alebo organizáciami (t. j. môže ju používať každý, kto má o to záujem). Vyššie uvedené štandardy (XML, XSD, SOAP, HTTP, MIME a WSDL) patria medzi otvorené štandardy.

Okrem ôsmich základných princípov SOA existujú aj **d'alšie dôležité charakteristiky SOA**, ako sú napr.:

1. *SOA nie je viazaná na konkrétnu technologickú platformu* – používanie SOA nie je viazané na vlastníctvo hardvéru a softvéru konkrétej značky alebo presne vymedzeného typu.
2. *SOA je nástrojom zjednodušujúcim integráciu aplikácií vo vnútri podniku, resp. organizácie, ale aj naprieč viacerými podnikmi, resp. organizáciami* – integrácia aplikácií je jednoduchšia najmä kvôli štandardizácii, a tiež kvôli tomu, že jednotlivé aplikácie reprezentované webovými službami nie sú pevne zviazané s konkrétnymi technologickými platformami.
3. *SOA nie je prekážkou rozšíriteľnosti systému* – v prípade potreby je do informačného systému na báze SOA možné pridávať nové služby alebo zlúčovať niektoré služby s inými. Je to možné s vykonávať len s minimálnym negatívnym dopadom na ostatné služby.
4. *SOA zvyšuje flexibilitu podnikov, resp. organizácií* – SOA nie je prekážkou pri snahách o reorganizáciu podnikov, resp. organizácií, o zlúčenie (t. j. fúzii) viacerých podnikov do jedného, alebo pri snahách o zmenu poľa pôsobnosti podniku. Pridávanie nových služieb a odstraňovanie starších je pri SOA pomerne jednoduché, a preto je možné informačný systém podniku alebo organizácie flexibilne prebudovať podľa aktuálnych potrieb jeho manažmentu.
5. *SOA nemusí znamenať nahradenie všetkých starších aplikácií, používaných v informačnom systéme, novšimi aplikáciami* – ak sa v podniku používajú staršie aplikácie, ktoré sa osvedčili, a manažment podniku necíti potrebu nahradzať tieto aplikácie webovými službami na báze SOA, je možné ich v informačnom systéme ponechať. Ich začlenenie do SOA sa dá zrealizovať vytvorením špeciálnych webových služieb sprostredkovávajúcich komunikáciu týchto aplikácií s inými webovými službami alebo aplikáciami tak, ako to znázorňuje obr. 17. Týmto spôsobom môžeme staršie aplikácie (alebo ľubovoľné aplikácie podľa potreby) plnohodnotným spôsobom začleniť do servisne orientovaného prostredia tvoreného webovými službami, ktoré sú vybudované v súlade s princípmi SOA.



Obr. 17: Štandardizované prepojenie dvoch starších aplikácií, resp. spôsob ich začlenenia do systému tvoreného webovými službami [ERL09]

6. Efektivita systému sa môže zvýšiť bezstavovosťou služieb, pretože režiu spojení s manažmentom údajov o vzájomnej interakcii medzi službami môže zabezpečovať samostatná aplikácia. Vďaka tomu môžu tieto služby rýchlejšie vykonávať svoju prácu – vzájomná interakcia medzi dvoma ľubovoľnými softvérovými programami si vyžaduje sledovanie určitých s tým spojených údajov, keďže každá ďalšia interakcia medzi týmito aplikáciami môže vychádzať z výsledkov, resp. výstupov ich predošej interakcie. Služba si tak musí vytvárať história svojej komunikácie s inými službami a evidovať doteraz dosiahnuté výsledky. Pri riešení zložitých problémov však môže byť výhodné manažment údajov o vzájomných interakciách medzi službami alebo interakciach medzi službami a ich koncovými používateľmi presunúť na samostatné aplikácie/služby (príkladom je už spomínaná orchestrácia ako model kompozície služieb, v rámci ktorej má centrálny koordinátor prehľad o stave riešenia problému a o čiastkových výsledkoch). Tým dochádza k odbremenneniu služieb od starostí s manažmentom údajov o ich interakcii s okolím. Dôsledkom je, že tieto služby môžu rýchlejšie vykonávať činnosti, ktoré sú ich skutočnou úlohou [ERL09].

S prechodom na SOA sú však spojené aj určité nevýhody, a to:

1. Vysoké počiatočné náklady – prebudovanie existujúceho informačného systému na SOA môže byť v závislosti od rozsahu a počtu potrebných aplikácií značne finančne nákladné. V záujme dosiahnutia čo najlepších výsledkov je vhodné ešte pred nasadením SOA vykonať analýzu podnikových procesov a jednotlivé služby navrhnuť tak, aby slúžili na čo možno najväčšiu podporu týchto procesov bez zbytočných duplicit a funkcionality, ktoré nie sú treba.

2. *Možnosť vzniku nevhodného konečného riešenia v dôsledku nedostatku znalostí o SOA*
  - komunikácia medzi službami tvoriacimi SOA je súčasťou standardizovaná, no napriek tomu neexistuje dostatočný počet odborných prác detailne opisujúcich „best practices“ v oblasti nasadzovania SOA. Neexistujú jednoznačné návody alebo príručky s dostatočnou detailnosťou na efektívnu implementáciu SOA, a preto si každá spoločnosť musí vytvoriť vlastný plán zavádzania SOA.
3. *Vysoké nároky na vybudovanie spoľahlivej a rýchlej IT infraštruktúry* – SOA je založená na vytvorení veľkého množstva služieb, ktoré spolu komunikujú prostredníctvom správ prenášaných pomocou počítačovej siete. Je preto potrebné počítať s pomerne vysokým zaťažením siete, ktorá musí byť vybudovaná tak, aby toto zaťaženie zvládla a zároveň zbytočne nespomaľovala prevádzku informačného systému.

Servisne orientovaná architektúra predstavuje jeden z prístupov k vytváraniu informačných systémov a k systémovej integrácii. **Systémová integrácia** je v informatike dôležitý pojem, ktorý Závodný definuje takto: „*Systémová integrácia sa chápe ako komplex činností smerujúcich k integrácii všetkých informačných podsystémov organizácie a služieb externých dodávateľov do integrovaného informačného systému podniku.*“ [ZÁV10]. Pojem **integrácia** pritom treba chápať ako *zjednocovanie, resp. prepájanie viacerých menších komponentov do jedného zmysluplného celku alebo ako cieľavedomé začleňovanie viacerých menších komponentov do jedného celku, čím sa tento celok postupne rozrastá*. Systémová integrácia je zložitá činnosť, ktorá pozostáva z mnohých čiastkových činností. Medzi ne patrí napr. *údajová integrácia*, čo je úprava údajov, pri ktorej sa odstraňujú rozdiely vo formátoch a údaje z rozličných zdrojov sa nahrávajú do jedného celku (databázy), aby sa s nimi dalo zmysluplnie pracovať a aby ich mohli „zdieľať“<sup>3</sup> všetky aplikácie a používateľia, ktorí k nim potrebujú prístup. Patrí sem tiež *integrácia aplikácií*, ktorej cieľom je zabezpečiť, aby boli všetky aplikácie v informačnom systéme, ktoré to potrebujú, schopné vzájomnej spolupráce a výmeny údajov, či *integrácia podnikových procesov*, ktorej cieľom je zasa zabezpečiť, aby podnikové procesy neprebiehali izolované a chaoticky (t. j. bez sledovania ich vzájomných nadvázností), ale aby dohromady vytvárali jeden zmysluplný a usporiadany komplex procesov, čím sa výrazne zvyšuje efektivita celého podniku.

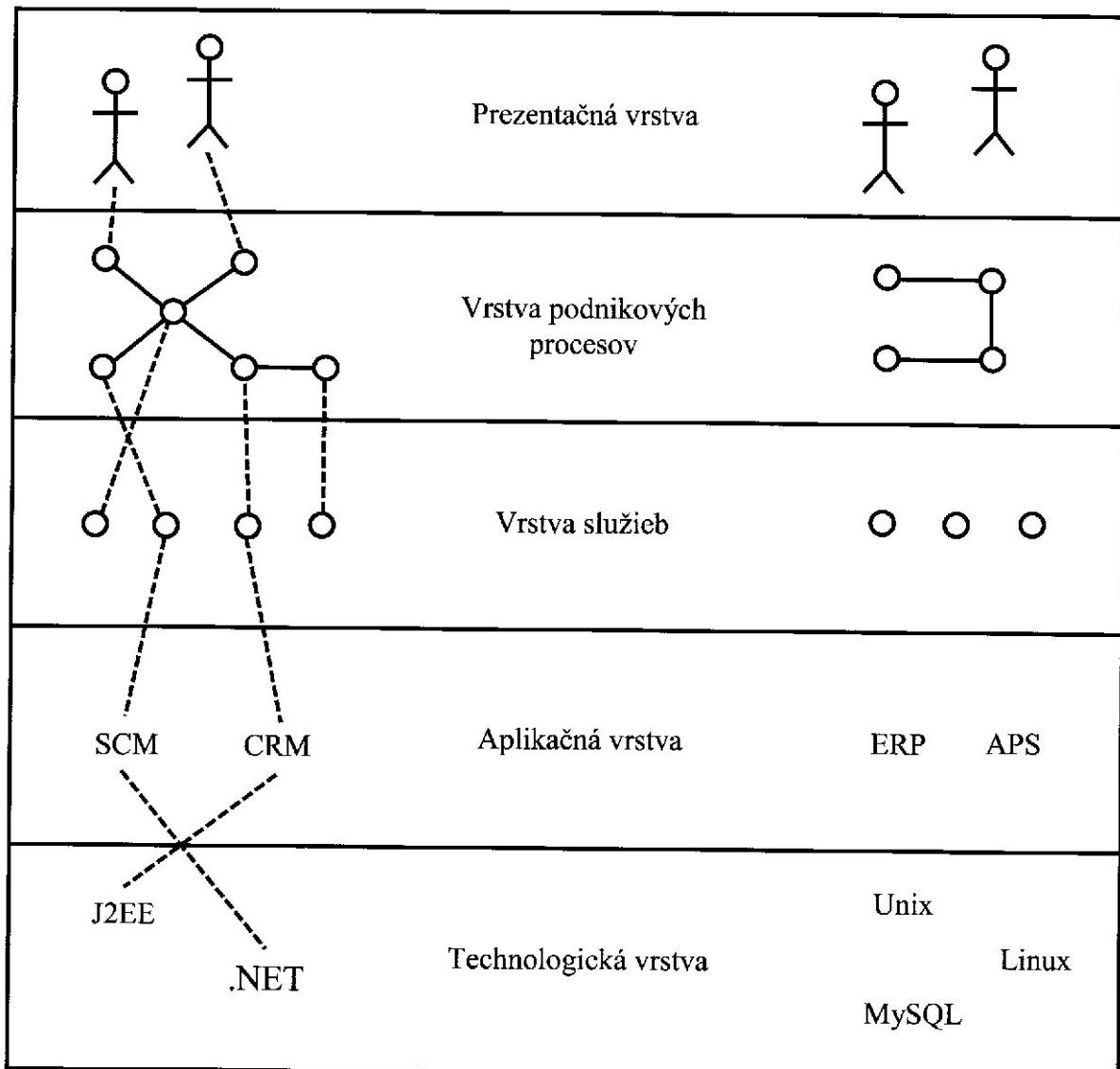
---

<sup>3</sup> Slovo „zdieľať“ je súčasťou hovornej terminológie, no doposiaľ nebolo oficiálne pridané do slovenského jazyka, a preto je uvedené v úvodzovkách. Slovenským ekvivalentom je slovné spojenie deliť sa o niečo, resp. podeliť sa o niečo.

Štumpf uvádzá zovšeobecnený referenčný model SOA, o ktorom piš aj Závodný vo svojej publikácii *Distribuované spracovanie dát*. Tento model pozostáva z piatich vrstiev, pričom v smere zhora nadol ide o tieto vrstvy [ZÁV10 a ŠTU06]:

- *Prezenčná vrstva* – ide o používateľské rozhranie (GUI), ktoré používateľom umožňuje prístup do systému a k jednotlivým jeho službám, a to (v závislosti od potrieb a možnosti toho-ktorého podniku) kdekoľvek, kedykoľvek a z akéhokoľvek zariadenia (teda nezávisle od výrobcu a konkrétnej značky).
- *Vrstva podnikových procesov* – táto vrstva je tvorená všetkými podnikovými procesmi, ktoré v tom-ktorom podniku prebiehajú, a umožňuje pospájať služby z nižzej vrstvy takým spôsobom, aby to vyhovovalo potrebám jednotlivých podnikových procesov – so snahou o čo najväčšiu znovupoužiteľnosť jednotlivých služieb (pokiaľ sa dá niektorá služba použiť na zabezpečenie potrieb viacerých podnikových procesov, je to vítané, lebo sa tým redukuje celkový počet služieb v systéme, čo systém zjednoduší). Na tejto vrstve sa teda rieši problematika vzájomnej kooperácie služieb vo vzťahu k potrebám podnikových procesov – *orchestrácia a choreografia*.
- *Vrstva služieb* – táto vrstva je tvorená všetkými službami, ktoré sú súčasťou systému. Jednotlivé typy služieb bližšie opíšeme v kapitole 3.3.
- *Aplikačná vrstva* – skutočnosť, že systém je vyskladaný z webových služieb, ncznamená, že jeho súčasťou nemôžu byť aj tradičné podnikové aplikácie, ako sú napr. aplikácie typov ERP, CRM, SCM, MES, APS či iné. Viac informácií o týchto a iných typoch podnikových aplikácií sa dá dočítať v [JUR18]. Webové služby môžu tieto aplikácie prepájať, poskytovať im prístup k údajom alebo inak podporovať ich činnosť.
- *Technologická vrstva* – ide o súhrn všetkých technológií, ktoré sú potrebné na chod všetkých aplikácií systému, na uchovávanie údajov, prácu s nimi a pod. Medzi komponenty technologickej platformy informačného systému môžeme vo všeobecnosti zaradiť najmä:
  - *hardvérové komponenty* – napr. servery, dátové úložiská, koncové stanice (počítače alebo iné zariadenia určené na prístup do systému), počítačové siete a pod.,
  - *softvérové komponenty* – napr. operačné systémy, databázové systémy, komunikačné systémy, softvér pre systémovú integráciu a pod.

Referenčný model SOA je schematicky znázornený na obr. 18.



Obr. 18: Referenčný model SOA [ZÁV10 a ŠTU06]

## Zhrnutie kapitoly 2

Pri funkčnom prístupe k riadeniu podniku je hlavnou úlohou IS podporovať vykonávanie funkcií jednotlivých organizačných útvarov. Pri procesnom riadení podniku je hlavnou úlohou IS podporovať správny priebeh podnikových procesov. SOA je jedným z prístupov, pomocou ktorých sa dajú vytvárať nové informačné systémy, ale aj prepájať už existujúce. Hlavnou myšlienkou je rozdeliť systém na menšie funkčné jednotky nazývané služby. Z technického hľadiska ide o webové služby, ktoré sú postavené na otvorených standardoch. Otvorený štandard v IT je publikovaná (t. j. zverejnená) špecifikácia určitého IT riešenia spravidla vytvorená štandardizačnou organizáciou, ktorá netrvá na žiadnych právnych obmedzeniach pri používaní tejto špecifikácie inými firmami alebo organizáciami (t. j. môže ju používať každý, kto má o to

záujem). Takýmito štandardmi sú napr. XML, XSD, WSDL, SOAP a iné. Služby sú autonómne jednotky schopné vzájomnej spolupráce. Tá môže prebiehať na princípe orchestrácie (spoluprácu medzi službami sprostredkúva centrálna koordinačná služba, ktorá ostatným zúčastneným službám zasiela inštrukcie a spúšťa ich v správnom poradí) alebo na princípe choreografie (spolupráca prebieha bez centrálnej koordinačnej služby).

## Literatúra ku kapitole 2

- [CLA91] CLAUS, V. – SCHVILL, A.: *Lexikón informatiky*. Bratislava: Slovenské pedagogické nakladateľstvo, 1991. 544 s. ISBN 80-08-00755-9.
- [ERL09] ERL, T.: *SOA Servisně orientovaná architektura*. 2009. Brno: Computer Press. 672 s. ISBN 978-80-251-1886-3.
- [GAR18] Gartner, Inc.: *Big Data*. In: *gartner.com* [online]. 2018. [Cit. 15. 2. 2020]. Dostupné na internete: <<https://www.gartner.com/it-glossary/big-data>>.
- [JUR18] JURÍK, P.: *Informačné systémy v podnikovej praxi*. 2018. 2. vyd. Nové Zámky: Tlačiareň MERKUR, s. r. o. 186 s. 978-80-970233-7-9.
- [KOK02] KOKLES, M. – ROMANOVÁ, A.: *Informačný vek*. 2002. Bratislava: Vydatelstvo Sprint vfra. 305 s. ISBN 80-89085-09-1.
- [MAN11] MANAGEMENTMANIA.COM: *Manažér (Manager)*. In: *managementmania.com* [online]. 2011. [Cit. 11. 7. 2020]. Dostupné na internete: <<https://managementmania.com/sk/manazer-manager>>.
- [NÁV07] NÁVRAT, P. a kol.: *Umelá inteligencia*. 2007. Bratislava: Vydatelstvo STU. 393 s. ISBN 978-80-22726-29-0.
- [SOA17] SOA: *Principles of Service Design Poster*. In: *soapatterns.org* [online]. 2017. [Cit. 31. 8. 2017]. Dostupné na internete: <[soapatterns.org/files/SOA\\_Principles\\_Poster.pdf](http://soapatterns.org/files/SOA_Principles_Poster.pdf)>.
- [SUJ14] SUJA, R. (Infostat): *Big data*. In: *infostat.sk* [online]. 2014. [Cit. 15. 2. 2020]. Dostupné na internete: <[http://www.infostat.sk/web2015/sk/\\_publikacie/Big\\_Data.pdf](http://www.infostat.sk/web2015/sk/_publikacie/Big_Data.pdf)>.
- [ŠTU06] ŠTUMPF, J: *Proč SOA nemá alternatívnu*. In: e-Focus, 4/2006, roč. 6, s 10 – 13.
- [ZÁV06] ZÁVODNÝ, P.: *Riadenie projektov informačných systémov*. 2006. Bratislava: Vydatelstvo Ekonóm. 166 s. ISBN 80-225-2230-9.

## Kapitola 3

### Základné charakteristiky webových služieb

V tejto kapitole charakterizujeme webové služby, ktoré sú základnou stavebnou jednotkou systémov vybudovaných na princípoch SOA. Po objasnení základných pojmov a faktov prejdeme na klasifikáciu webových služieb podľa potrieb SOA a na záver kapitoly uvedieme príklad, ktorý znázorňuje, ako sa dajú jednotlivé typy, resp. modely služieb využiť pri automatizácii konkrétneho podnikového procesu.

#### 3.1 Webové služby – základné fakty

*Webovú službu môžeme definovať ako rozhranie k určitej aplikácii bežiacej na serveri, ktoré umožňuje iným aplikáciám, aby ju mohli spustiť – zadávať do nej vstupy a prijímať výstupy, a to prostredníctvom štandardných komunikačných protokолов, akými sú najmä HTTP, resp. HTTPS a SOAP. Webová služba teda zabezpečuje prístup k určitej naprogramovanej funkcií alebo kollekcií funkcií, ktoré spolu súvisia a majú jasne špecifikovanú funkciaľitu, vstupy a výstupy. Všimnime si, že v tejto definícii sa nehovorí o komunikácii ako o výmene údajov a inštrukcií medzi aplikáciou a jej používateľmi, ale len medzi aplikáciou a inými aplikáciami.*

Organizácia pre rozvoj štruktúrovaných informačných štandardov (The Organization for the Advancement of Structured Information Standards, OASIS) definuje webovú službu ako „mechanizmus zabezpečujúci prístup k jednej alebo viacerým funkciaľitám s tým, že tento prístup sa realizuje prostredníctvom rozhrania služby pri dodržaní všetkých obmedzení a pravidiel, ktoré sú špecifikované v opise služby“ [OAS06]. Služby bývajú zverejnené v registri, ktorý sa označuje ako UDDI (Universal Description, Discovery and Integration). Tento register umožňuje tvorcom služieb, aby v ňom zaregistrovali svoje služby a zverejnili ich pre potenciálnych záujemcov o ich používanie. Záujemcovia zasa môžu prostredníctvom tohto registra vyhľadať službu, ktorá najviac vyhovuje ich požiadavkám, a spustiť ju tak, že jej zašlú správu vo formáte SOAP prostredníctvom prenosového protokolu HTTP (najčastejšie sa používa HTTP, ale dajú sa využiť aj iné protokoly). Registre UDDI môžu byť verejné alebo súkromné a viac o nich uvedieme v kapitole 6.4.

Návrh a tvorba informačných systémov na báze SOA zvyšujú interoperabilitu (t. j. schopnosť systémov vzájomne si poskytovať služby a efektívne spolupracovať) medzi organizačnými jednotkami (t. j. prvkami organizačnej štruktúry, podniku, resp. organizácie, pre ktorú informačný systém navrhujeme) s cieľom zabezpečiť spoločný prístup k informáciám

(nepisovne „zdieľanie informácií“). Ak každé oddelenie sprístupní pre iné oddelenia služby, ktoré zabezpečujú prístup k jeho vybraným informáciám, potom si iné oddelenia môžu ľahko vypýtať informácie, ktoré potrebujú pre svoju činnosť. Ak vznikne nové oddelenie, je pomerne ľahké vytvoriť pre neho webovú službu na správu jeho informácií a túto službu začleniť do systému ostatných služieb používaných v informačnom systéme. Naopak, ak jednotlivé oddelenia v podniku nepoužívajú webové služby založené na štandardoch, ale používajú tradičné podnikové aplikácie pochádzajúce od rozličných výrobcov, ktoré sú naprogramované v rozličných jazykoch alebo musia byť prevádzkované na rozličných technologických platformách, môže to výrazne komplikovať zabezpečovanie výmeny údajov a inštrukcií medzi týmito aplikáciami, a teda aj zodpovedajúcimi oddeleniami.

Ako sme už naznačili, webové služby sú primárne založené na komunikácii typu stroj-stroj, t. j. na vzájomnú komunikáciu medzi webovými službami (aplikáciami). Tradičné webové služby preto vôbec nemávajú **používateľské rozhranie**, t. j. rozhranie určené na výmenu údajov medzi aplikáciou a jej používateľom (človekom), resp. na prenos inštrukcií smerom od používateľa k aplikácii. Namiesto používateľského rozhrania mávajú webové služby tzv. API (Application Program Interface), t. j. rozhranie určené na komunikáciu s inými aplikáciami. Toto rozhranie umožňuje, aby určitá aplikácia (služba) mohla prijať správu od inej aplikácie v standardizovanej forme (napr. XML, SOAP alebo JSON – pri SOA sa však najčastejšie používajú správy formátované v súlade s pravidlami protokolu SOAP) a aby mohla v standardizovanej forme zasielať správy iným aplikáciám (službám).

Absencia používateľského rozhrania je jedným z hlavných *rozdielov medzi webovou službou a webovou aplikáciou*. **Webová aplikácia** je aplikácia, ktorá je spustená na vzdialenom fyzickom serveri a používateľ k nej pristupuje pomocou webového prehliadača (teda klientskej aplikácie) na svojom fyzickom zariadení (počítač, tablet, telefón a pod.). K aplikácii sa môžu pripojiť viacerí používatelia so žiadostou o poskytnutie určitých údajov alebo o vykonanie určitej aktivity, a server, na ktorom je aplikácia spustená, obslúži ich požiadavku. Webové aplikácie sú teda postavené na komunikácii typu *klient-server*, t. j. viaceré klientske aplikácie sa podľa potreby obracajú na aplikáciu, ktorá zo softvérového hľadiska vystupuje ako server. Tá im poskytuje určitú službu. Aplikácia, ktorá je spustená na fyzickom serveri, vystupuje aj zo softvérového hľadiska vždy ako server, pretože iba reaguje na požiadavky prichádzajúce od klientov a obsluhuje ich. Webové prehliadače v tomto vzťahu zase zo softvérového hľadiska vystupujú vždy iba v pozícii klientov, pretože slúžia iba na zadávanie požiadaviek a nikdy nie na ich vykonanie, resp. obslúženie.

Naproti tomu, každá **webová služba** môže zo softvérového hľadiska pôsobiť ako *server*, ale aj ako *klient*. To znamená, že má možnosť obrátiť sa na iné služby so žiadostou o poskytnutie určitých údajov alebo o vykonanie určitej činnosti (vtedy voči týmto službám vystupuje ako *klient*), ale zároveň aj iné služby majú možnosť obrátiť sa na ňu so žiadostou o poskytnutie určitých údajov alebo vykonanie určitej činnosti (vtedy voči týmto službám vystupuje ako *server*). Služby, ktoré spolu takýmto spôsobom zo softvérového hľadiska komunikujú ako klient a server, môžu byť z hardvérového hľadiska prevádzkované na vzdialených fyzických serveroch, no môžu byť prevádzkované aj na jednom a tom istom fyzickom serveri. Na jednom fyzickom zariadení môže byť totiž naraz prevádzkovaných viacero webových služieb, ktoré sa potom musia vzájomne deliť o jeho výpočtové prostriedky (operačnú pamäť, výkon procesora, prístup k databáze atď.).

SOA sa preto hodí skôr na projektovanie a tvorbu systémov, pri ktorých sa predpokladá, že bude dochádzať v prevažnej miere ku vzájomnej komunikácii medzi aplikáciami a nebude vo veľkom rozsahu dochádzať k priamej interakcii medzi aplikáciami a ich používateľmi. Na tvorbu aplikácií, ktoré vykazujú vysokú mieru interaktivity (t. j. používateľ má veľa možností zasiahnuť do výkonu ich činnosti), sa preto SOA príliš nehodí.

Ako sme už uviedli, v prostredí tvorenom webovými službami môže každá služba vystupovať zo softvérového hľadiska voči ostatným službám niekedy v pozícii *klienta*, inokedy ako *server* v závislosti od kontextu. Úloha, resp. rola konkrétnej služby ako servera či klienta sa môže v priebehu riešenia zložitejších úloh niekoľkokrát zmeniť (raz táto služba požaduje vykonanie určitej akcie od inej služby, t. j. vystupuje vo vzťahu k nej ako klient, inokedy sa priamo od nej požaduje vykonanie určitej akcie, a teda je táto služba postavená do pozície servera).

Predstavme si napr., že služba A sa obracia na službu B so žiadostou o poskytnutie určitých údajov, no služba B nie je schopná všetky požadované údaje získať sama a musí sa preto obrátiť na službu C so žiadostou o poskytnutie chýbajúcich údajov. Vo vzťahu k službe A vystupuje služba B ako server (pretože jej úlohou je „obslúžiť“ službu A), no vo vzťahu k službe C vystupuje služba B ako klient (pretože od služby C požaduje poskytnutie určitého výkonu). Tento príklad ilustruje skutočnosť, že určitá služba môže vo vzťahu k niektorým službám vystupovať ako klient, no vo vzťahu k iným ako server – vždy v závislosti od kontextu, v akom dochádza k interakcii s inými službami.

Webová služba prijíma rolu *poskytovateľa služieb* (t. j. rolu *servera*) za týchto podmienok:

- je volaná externým zdrojom, akým je napr. *žiadateľ služieb* (t. j. *klient*),
- poskytuje opis služby, ktorý obsahuje informácie o jej funkciách a správaní.

Rola poskytovateľa služieb je ekvivalentná s rolou servera v klasickej architektúre klient/server. Termín *poskytovateľ služieb* sa však používa aj na označenie spoločnosti alebo jednotlivca, zodpovedného za sprístupnenie a prevádzku webovej služby. Aby bolo možné tieto dva významy od seba lepšie odlišiť, používajú sa niekedy nasledujúce presnejšie termíny:

- *entita poskytovateľa služieb* – označenie pre spoločnosť alebo jednotlivca sprístupňujúceho a prevádzkujúceho webovú službu,
- *agent poskytovateľa služieb* – označenie pre konkrétnu webovú službu poskytovanú určitou spoločnosťou alebo jednotlivcom.

Najčastejšie sa však pre volanú službu používa označenie poskytovateľ služieb, pretože termín agent poskytovateľa služieb je zbytočne zložitý [ERL09]. Ako uvádza Erl: „Akákoľvek jednotka logiky spracovania schopná vydať správu s dopytom, ktorému rozumie poskytovateľ služieb, je klasifikovaná ako žiadateľ služieb. *Webová služba je vždy poskytovateľom služieb, ale môže vystupovať tiež ako žiadateľ služieb.*“ [ERL09].

Webová služba prijíma rolu *žiadateľa služieb* (t. j. *klienta*) za nasledujúcich podmienok:

- hľadá a identifikuje najlepšieho poskytovateľa služieb na základe dostupných opisov služieb,
- volá poskytovateľa služieb odoslaním správy,
- ak od poskytovateľa služieb požaduje zasланie spätej správy s výsledkom, musí sprístupniť svoj opis, aby poskytovatelia služieb vedeli, v akom tvare má byť zapísaná táto správa, a tiež kam a akým spôsobom ju treba poslať.

Rola žiadatelia služieb je ekvivalentom roly klienta v klasickej architektúre klient/server. Významovo je termín *žiadateľ služieb* opäť dvojzmyselný a môže označovať jednak konkrétnu webovú službu, ale aj jej vlastníka. Existujú preto dva upresňujúce pojmy, ktoré však nie sú vo veľkej miere používané:

- *entita žiadateľa služieb* – označenie pre spoločnosť alebo jednotlivca žiadajúceho webovú službu,

- *agent žiadateľa služieb* – označenie pre samotnú webovú službu, ktorú určitá spoľnosť alebo jednotlivec požaduje.

Komunikácia webových služieb formou odosielania správ je založená na použití ciest pre výmenu správ. Tieto cesty nemusia byť vždy priame (t. j. správa nemusí byť vždy smerovaná *priamo od žiadateľa služby k jej poskytovateľovi*, ale môže prechádzať cez jednu alebo viacero medziľahlých služieb, ktoré rozhodnú o správnom smerovaní tejto správy tak, aby bola v konečnom dôsledku doručená správnemu poskytovateľovi služby, zabezpečujúcemu obslúženie požiadavky). Ako uvádzajú Erl: „*webové služby, ktoré smerujú a spracovávajú správu po jej odoslani a predtým, než je doručená do konečného cieľa, sa dajú označiť ako sprostredkovatelia alebo sprostredkujúce služby. Pretože sprostredkovateľ prijíma a odosielá správy, strieda role poskytovateľa služieb a žiadateľa služieb.*“ [ERL09].

Poznáme *dva typy sprostredkovateľov* [ERL09]:

- *Pasívni sprostredkovatelia* – zodpovedajú za smerovanie správ do ďalších uzlov komunikačnej siete (teda k iným službám). Pri smerovaní môžu využiť informácie nachádzajúce sa v hlavičke SOAP správy (t. j. priložené metadáta), ale môžu tiež použiť vlastnú logiku smerovania. Pre pasívneho sprostredkovateľa je charakteristické to, že nijakým spôsobom neupravuje správu, len ju posúva ďalej.
- *Aktívni sprostredkovatelia* – ich úlohou je takisto smerovať správy od ich odosielateľa smerom k cieľovej službe, no na rozdiel od pasívnych sprostredkovateľov tieto správy pred odoslaním spracujú a pozmenia, resp. upravia ich obsah. Zvyčajne sa takito sprostredkovatelia pozrú na určité bloky v hlavičke SOAP správy (metadáta) a vykonajú nejakú akciu podľa informácií, ktoré v nej nájdú. Najčastejšie ide o zmeny v niektorých konkrétnych blokoch hlavičiek s tým, že môžu dokonca takéto bloky do správy pridávať alebo ich aj mazat’.

V rámci cesty, po ktorej putuje správa, označujeme toho žiadateľa služieb, ktorý je počiatočným iniciátorom prenosu tejto správy, ako *počiatočného odosielateľa*. Ide teda vždy o prvú webovú službu na ceste konkrétnej správy. Ako *konečného prijemcu* zasa označujeme poskytovateľa služieb, ktorý je poslednou webovou službou na ceste tejto správy. Sprostredkovatelia (aktívni a ani pasívni) teda nikdy nemôžu byť v pozícii počiatočného odosielateľa alebo konečného prijemcu správy.

### **3.2 Webové služby a používateľské rozhranie**

Napriek tomu, že webové služby nie sú určené na priamu interakciu s človekom, je často potrebné takúto interakciu zabezpečiť. Každý informačný systém totiž musí v prvom rade slúžiť svojim používateľom, a to platí aj o systémoch vyskladaných z webových služieb. Niektoré webové služby môžu byť naprogramované tak, že potrebujú prijať určité vstupné údaje alebo inštrukcie od používateľa a až na základe nich môžu konať. To znamená, že musí existovať **používateľské rozhranie**, prostredníctvom ktorého môže používateľ webovej službe zadávať inštrukcie, poskytovať jej vstupné údaje a prijímať od nej výstupy. Na zadávanie vstupných údajov sa v podnikovom prostredí zväčša používajú graficky nenáročné formuláre.

Používateľské rozhranie môžeme chápať ako softvérovú vrstvu medzi používateľom a určitou systémovou funkcionalitou. V systéme vybudovanom na báze SOA býva funkcionalita distribuovaná. To znamená, že na obslúženie požiadavky používateľa (alebo inej aplikácie) môže byť potrebná spolupráca viacerých aplikácií (webových služieb), ktoré sú prevádzkované na rozličných uzloch (fyzických serveroch) počítačovej siete. Z tohto dôvodu používateľské rozhrania v SOA nebývajú tesne zviazané s aplikáciami (službami), ku ktorým zabezpečujú prístup. Používajú sa najmä tieto formy rozhraní [SEN09]:

1. *Tenki klienti* – na zariadení používateľa musí byť nainštalovaná aplikácia, ktorá plní iba úlohu používateľského rozhrania a nezabezpečuje žiadnu aplikačnú logiku ani prácu s databázou. Takoto aplikáciou býva najčastejšie webový prehliadač. Používateľské rozhranie je vtedy reprezentované webovou stránkou, ktorú si používateľ zobrazí vo svojom prehliadači a môže s ňou pracovať. Na zariadení používateľa je teda prevádzkované len používateľské rozhranie (jeho webový prehliadač), zatiaľ čo spracovanie údajov a fyzická práca s databázou prebiehajú na vzdialených serveroch dostupných prostredníctvom internetu.
2. *Podnikové portály* – podnikový portál je „*množina technológií a aplikácií tvoriacich univerzálné rozhranie, ktorého prostredníctvom je každému, koho sa týkajú činnosti podniku, resp. organizácie, umožnené zúčastniť sa na procesoch tohto podniku, resp. organizácie, pristupovať ku všetkým relevantným informáciám, komunikovať s ostatnými kooperujúcimi pracovníkmi a realizovať adekvátne aktivity súvisiace s podnikovými procesmi*“ [POU15]. Ide teda o integráciu aplikácií na úrovni používateľského rozhrania. Zvykne sa realizovať tak, že sa vytvorí jedno prostredie, do ktorého sa používatelia prihlasujú pomocou používateľského mena a hesla).

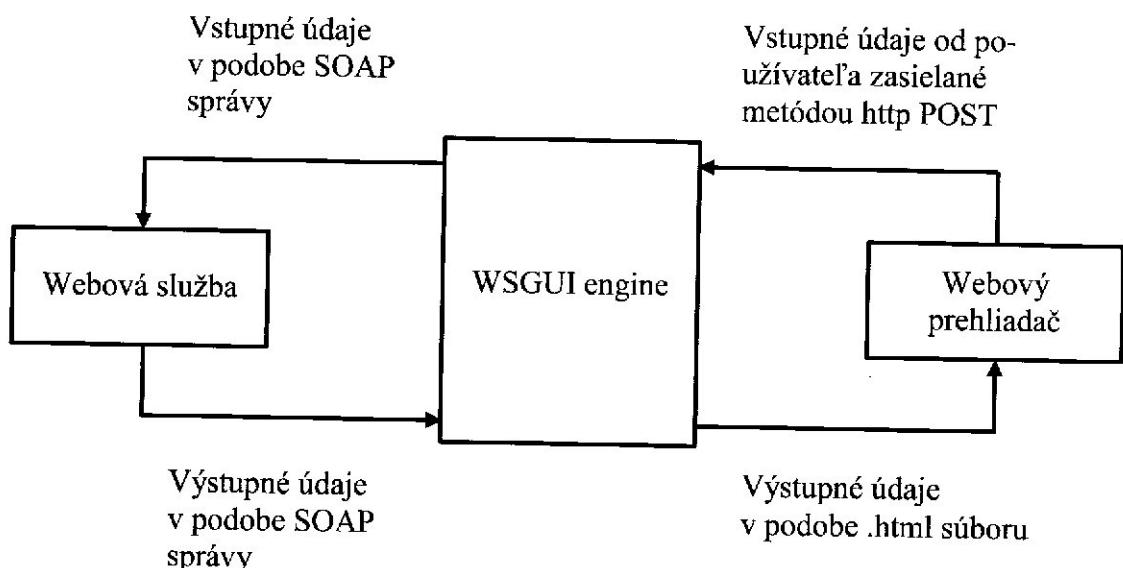
Po prihlásení môžu pristupovať ku všetkým službám, resp. aplikáciám v informačnom systéme. Pritom sa musia zohľadňovať prístupové práva jednotlivých používateľov, t. j. každý používateľ smie mať prístup iba k tým aplikáciám, ktoré skutočne potrebuje pre svoju prácu. Používateľ pracuje s podnikovým portálom pomocou svojho webového prehliadača.

3. *Tuční klienti* – na zariadení používateľa musí byť nainštalovaná špeciálna *aplikácia*, ktorá sa spúšťa v samostatnom okne (mimo webového prehliadača). Vďaka tejto aplikácii dochádza k spracovaniu údajov priamo na zariadení používateľa a v prípade potreby sa aplikácia môže obrátiť na vzdialený server so žiadosťou o prístup k databáze (za účelom zápisu, čítania, modifikácie alebo mazania údajov) alebo o realizáciunejakej špecifickej operácie. Tuční klienti sa zvyknú používať vo firemnom, resp. organizačnom prostredí v rámci lokálnej počítačovej siete (LAN).
4. *Chytrí klienti* – ide o riešenie, ktoré kombinuje vybrané rysy tenkých a tučných klientov. Na zariadení používateľa sa nainštaluje aplikácia, ktorá sa spúšťa v samostatnom okne (mimo webového prehliadača). Táto aplikácia uskutočňuje väčšinu operácií priamo na zariadení používateľa (t. j. toto zariadenie musí znášať väčšinu záťaže súvisiacej so spracovaním údajov), no v prípade potreby sa môže prostredníctvom internetu pripojiť na vzdialé servery so žiadosťou o prístup k databáze alebo o realizáciu určitej špecifickej operácie (t. j. aj server býva do určitej miery zaťažený spracovaním údajov). Tieto servery sa pritom nemusia nachádzať v lokálnej počítačovej sieti (ako je to typické pre *tučných klientov*), ale môže ísť o ľubovoľné vzdialé servery dostupné cez internet.

Každý z vyššie uvedených štyroch spôsobov realizácie používateľského rozhrania umožňuje výmenu údajov medzi používateľom a webovými službami v systéme vytvorenom na princípoch SOA. Pritom však musí byť dodržaná zásada, podľa ktorej webové služby vzájomne komunikujú prostredníctvom zasielania štandardizovaných správ (najčastejšie správ vo forme SOAP posielaných pomocou protokolu HTTP), a to platí aj o komunikácii medzi službami a ich používateľmi. *Používateľské rozhranie preto musí byť schopné transformovať vstupné údaje a inštrukcie získané od používateľa do podoby SOAP správy a túto správu poslať tej službe, ktorej sú určené*. Ak to používateľské rozhranie nie je schopné urobiť (napr. webové prehliadače nie sú schopné pracovať so SOAP správami), potom sa medzi toto rozhranie a webovú službu musí vložiť medziľahlá softvérová vrstva (aplikácia) schopná transformovať vstupné údaje a inštrukcie zadané používateľom do používateľského rozhrania na SOAP správu

a túto správu poslať cieľovej webovej službe. Takáto medziľahlá softvérová vrstva sa zvykne označovať ako **WSGUI engine**.

Ak má webová služba po spracovaní vstupných údajov a inštrukcií od používateľa poslať používateľovi nejaké výstupné údaje, potom tieto údaje zapíše do SOAP správy, ktorú pošle vrstve WSGUI engine. *WSGUI engine SOAP správu transformuje do podoby akceptovateľnej pre používateľské rozhranie* (ak je používateľským rozhraním webový prehliadač, potom túto správu transformuje na .html súbor) *a v takejto podobe ju tomuto rozhraniu prepošle*. Rozhranie potom zobrazí výstup webovej služby používateľovi. Tieto vzťahy sú vizuálne znázornené na obr. 19.



Obr. 19: **Komunikácia medzi webovým prehliadačom a webovou službou sprostredkovana pomocou medziľahlej softvérovej vrstvy WSGUI engine** [Zdroj: autor]

### 3.3 Klasifikácia webových služieb podľa potrieb SOA

Rozličné spôsoby, akými sú webové služby využívané v rámci systémov ako ich súčasti, viedli k ich klasifikácii založenej na povahе aplikačnej logiky, ktorú zabezpečujú, a vyjadrujúcej ich rolu vo vzťahu k podnikovým procesom a čiastkovým úlohám (krokom) v rámci týchto procesov.

Podľa tejto klasifikácie rozlišujeme nasledujúce **modely služieb**:

1. **Entitné služby** – v každom podniku existujú významné entity, ktoré majú veľký vplyv na jeho podnikateľskú alebo ekonomickú činnosť. Za podnikovú entitu môžeme

považovať ľubovoľný objekt (živý aj neživý; resp. hocičo, čo jestvuje, a možno mať o tom údaje, resp. informácie). Typickými príkladmi podnikových entít sú najmä zákazník, dodávateľ, zamestnanec, objednávka, faktúra, pohľadávka, záväzok, výrobok, materiál, pobočka, sklad, výrobná hala a pod. S podnikovými entitami sa spája množstvo údajov, ktoré ich bližšie charakterizujú a ktoré sa zvyknú uchovávať v databáze. **Entitná služba** je služba, ktorá sa vzťahuje na konkrétnu podnikovú entitu a jej úlohou je pristupovať do databázy a zabezpečovať štyri operácie základného prístupu k údajom o tejto entite, známe pod skratkou **CRUD**:

- C (create) – zápis nových údajov o vybranej podnikovej entite do databázy,
- R (read) – prečítanie údajov z databázy,
- U (update) – modifikácia (t. j. úprava) údajov zapísaných v databázce,
- D (delete) – vymazávanie údajov z databázy.

Zamerajme sa napr. na entitu „Výrobok“. Je zrejmé, že výrobky sú dôležitými objektmi týkajúcimi sa podnikania, súvisí s nimi množstvo údajov, ktoré treba uchovávať v databáze, a preto ich právom môžeme považovať za podnikové entity. Predstavme si, že základné údaje o výrobkoch uchovávame tak, ako to zobrazuje nasledujúci úryvok tabuľky (tab. 5).

Tab. 5: Úryvok z tabuľky výrobkov [Zdroj: autor]

<i>ID_výrobku</i>	<i>Názov_výrobku</i>	<i>Jednotkové_náklady</i>	<i>Jednotk_predajná_cena</i>	...
101	Jogurt čokoládový	0,21 EUR	0,34 EUR	...
102	Jogurt čučoriedkový	0,23 EUR	0,36 EUR	...
103	Tvaroh odtučnený	0,36 EUR	0,43 EUR	...
...	...	...	...	...

V tejto tabuľke každý riadok reprezentuje iný typ výrobku a v jednotlivých stĺpcach sa nachádzajú údaje bližšie opisujúce výrobky v príslušných riadkoch. Údaje

v jednom riadku teda vždy významovo patria k sebe a vzťahujú sa na jeden typ výrobku. Operácia typu „*Create*“ znamená, že do tabuľky výrobkov pridáme nový výrobok a všetky s ním súvisiace údaje (resp. tie, ktoré poznáme) zapíšeme do jednotlivých buniek v tomto riadku. V tabuľke výrobok teda pribudne nový riadok. Operácia typu „*Read*“ znamená, že vyhľadáme konkrétny údaj, resp. množinu konkrétnych údajov o konkrétnom výrobku a poskytujeme ich používateľovi alebo aplikácii, ktorá ich požaduje. Operácia typu „*Update*“ znamená, že pristupujeme ku konkrétnemu údaju, resp. naraz k viacerým údajom v tabuľke pomocou jedného príkazu a zmeníme ich. Operácia typu „*Delete*“ znamená vymazávanie údajov – spravidla ide o vymazanie celého riadka tabuľky, čo môže pri kaskádových prepojeniach spôsobiť vymazanie súvisiacich záznamov aj v iných tabuľkách (tzv. „*Delete on cascade*“).

Entitné služby majú zvyčajne vysokú mieru znovupoužiteľnosti. To znamená, že nebývajú úzko späté len s jedným konkrétnym podnikovým procesom, ale zúčastňujú sa na priebehu viacerých podnikových procesov alebo podprocesov v konkrétnom podniku. V podniku môžu totiž prebiehať viaceré procesy, pri ktorých treba pristupovať k údajom o výrobkoch alebo k údajom o objednávkach, faktúrach a pod. S entitou „*Výrobok*“ napr. súvisí proces „*Nákup materiálu*“, proces „*Vybavovanie zákaznickej objednávky*“, proces „*Výroba*“ a všetky jeho podprocesy, proces „*Riešenie reklamácií a stážnosti zo strany zákazníkov*“, „*Preprava tovaru k zákazníkovi*“ a mnoho ďalších procesov. Významné podnikové entity teda spravidla súvisia s viac ako jedným podnikovým procesom a to platí aj o entitných službách, ktoré sa na tieto entity vzťahujú.

Príklad entitnej služby aj s názvami konkrétnych funkcií, ktoré táto služba ponúka (t. j. sprístupňuje pre iné služby), je uvedený na obr. 20. Touto službou je služba „*Výrobok*“, ktorej úlohou je pristupovať do databázy výrobkov a zabezpečovať vykonávanie štyroch základných databázových operácií (CRUD). To znamená, že ak potrebujeme vykonať ľubovoľnú zo štyroch vyššie uvedených operácií v súvislosti s údajmi o nejakom výrobku, môžeme sa kedykoľvek obrátiť na túto službu (t. j. poslať jej správu so zodpovedajúcou požiadavkou) a ona sa o to postará. Pre entitné služby je preto typické, že sa na ne obracajú služby iných typov, ak potrebujú údaje z databáz, do ktorých nemajú priamy prístup. Entitné služby im tento prístup sprostredkujú a poskytnú im požadované údaje. Podnet na prístup k údajom ale môže pochádzať aj od používateľa IS – v tom prípade sa tento podnet musí transformovať na SOAP správu so žiadostou o údaje, ktorá je určená konkrétnej entitnej službe.

Výrobok
<ul style="list-style-type: none"> <li>• Pridať_výrobok (<u>vstup</u>: údaje o novom type výrobku; <u>výstup</u>: nový riadok v tabuľke výrobkov)</li> <li>• Poskytnúť_údaje_o_výrobku (<u>vstup</u>: špecifikácia požadovaných údajov; <u>výstup</u>: údaje žiadane žiadateľom)</li> <li>• Zmeniť_údaj_o_výrobku (<u>vstup</u>: špecifikácia toho, ktorý údaj sa má zmeniť, a špecifikácia jeho novej hodnoty; <u>výstup</u>: zmenený údaj)</li> <li>• Vymazať_výrobok (<u>vstup</u>: špecifikácia riadka, ktorý sa má odstrániť – alebo spoločnej charakteristiky viacerých riadkov; <u>výstup</u>: odstránený riadok alebo riadky)</li> </ul>

Obr. 20: Entitná služba „Výrobok“ a funkcie, ktoré sprístupňuje svojmu okoliu  
[Zdroj: autor]

2. *Služby vzťahujúce sa na podnikové procesy* – ide o služby, ktoré nie sú úzko zviazané s jednotlivými podnikovými entitami, ale majú, naopak, úzku väzbu na konkrétné podnikové procesy. Ich úlohou je zabezpečovať automatizáciu niektorých činností, ktoré sa vykonávajú ako súčasť konkrétnych podnikových procesov. V dôsledku toho dochádza k odbremenению človeka od výkonu týchto činností a k zabezpečeniu plynulejšieho a rýchlejšieho chodu podnikových procesov. Služby viažuce sa na podnikové procesy môžeme rozčleniť do dvoch kategórií: **koordinačné služby a spracovateľské služby**.

**Koordinačné služby** sú v literatúre niekedy označované aj ako *inštrumentačné služby*, resp. *služby inštrumentácie*. Úlohou koordinačnej služby je koordinovať (to znamená usmerňovať) činnosť iných služieb, ktoré sú jej podriadené tak, aby sa všetky tieto služby spustili v správnom poradí, aby presne vedeli, čo majú robiť, a aby bol dosiahnutý požadovaný výsledok, vyplývajúci z ich kooperácie, resp. aby bol dodržaný správny tok (t. j. následnosť krokov) určitého podnikového procesu. Koordinačná služba musí teda usmerňovať množinu služieb, ktoré sú jej podriadené, a to tak, aby poradie ich spúšťania a ich činnosť boli v súlade s plánovaným priebehom konkrétneho

podnikového procesu alebo podprocesu a aby tento proces alebo podproces mohol byť dovedený do úspešného konca. Podriadenými službami môžu byť iné služby vzťahujúce sa na podnikové procesy (koordinačné alebo spracovateľské), ale môžu to byť tiež entitné a pomocné služby.

Koordinačné služby sa zvyčajne pomenovávajú v súlade s názvom podnikového procesu alebo podprocesu, ktorého logiku reprezentujú, resp. na ktorého automatizáciu slúžia (t. j. koordinačnú službu slúžiacu na automatizáciu určitého procesu alebo podprocesu je vhodné nazvať rovnako, ako sa volá tento proces alebo podproces).

Koordinačná služba je schematicky znázornená na obr. 21. Ide o službu slúžiacu na riadenie automatizácie podnikového procesu s názvom „*Riadenie objednávania materiálu*“, na ktorom sa spolupodieľajú štyri iné služby, od ktorých functionality a názov teraz pre jednoduchosť abstrahujeme. Táto koordinačná služba obsahuje iba jedinú funkciu s názvom „*Vykonaj*“, v ktorej je naprogramovaná všetka logika (t. j. všetky pravidlá) spúšťania jej podriadených služieb. Aby bola koordinačná služba schopná komunikovať so všetkými jej podriadenými službami, musí byť schopná postupne prijímať ich výstupy (údaje) a podľa potreby im poskytovať vo vhodnom okamihu vhodné vstupy (údaje a inštrukcie). K výmene všetkých údajov a inštrukcií medzi ňou a jej podriadenými službami dochádza prostredníctvom vhodne naformátovaných SOAP správ.

Riadenie objednávania materiálu	
Funkcia:	Vykonaj
Vstupy:	výstupné údaje všetkých podriadených služieb prichádzajúce priebežne počas vykonávania konkrétnej inštancie riadeného podnikového procesu
Výstupy:	vstupy postupne adresované podriadeným službám vo vhodnom čase vzhľadom na aktuálny priebeh a vývoj konkrétnej inštancie procesu

Obr. 21: **Koordinačná služba „Riadenie objednávania materiálu“** [Zdroj: autor]

Ako sme už spomenuli, okrem koordinačných služieb medzi služby vztahujúce sa na podnikové procesy patria aj tzv. **spracovateľské služby** (v literatúre sa pre ne niekedy používa pomerne komplikovaný pojem *služby riadenia vztahujúce sa k úlohe*). Spracovateľská služba má úzku väzbu na konkrétny podnikový proces konkrétneho podniku a súvisí so spracovaním údajov, t. j. s analýzou údajov, s rozličnými formami výpočtov, s porovnávaním, posudzovaním, vyhodnocovaním údajov a pod. Ide teda o rozličné výpočty, plánovanie rôznych aktivít pri zohľadnení stanovených podmienok a obmedzení, analýzy údajov za účelom potvrdenia nejakých hypotéz, predikcie budúceho vývoja (napr. spotrebiteľského správania na trhu, vývoja cien a pod.), identifikáciu vztahov, resp. závislostí medzi údajmi, tzv. korelácií (napr. ako súvisí predaj určitého výrobku s aktuálnym ročným obdobím alebo ako súvisí zmena spotrebiteľských preferencií s aktuálnym dianím v určitom regióne), zostavovanie rozličných typov reportov, dashboardov a dokumentov určených pre manažérov ako podklad pri ich rozhodovaní sa o dôležitých podnikových otázkach, alebo o realizácii iných činností, ktoré by inak v rámci zabezpečovania chodu konkrétnych podnikových procesov museli vykonávať ľudia.

Dôležité je, že spracovateľské služby sa vždy *priamo týkajú vnútorných pravidiel, postupov a aktivít podniku*, pre ktorý navrhujeme informačný systém. Mohli by sme ich teda nazvať aj *podnikovo orientované spracovateľské služby*, resp. **business služby**. To znamená, že v sebe majú *implementovanú určitú firemnú výpočtovú logiku alebo sa v nich odrážajú určité firemné postupy*. Na ich návrhu by sa preto mali zúčastňovať odborníci na podnikateľskú činnosť, resp. s tým súvisiace špecifické oblasti, akými sú napr. výroba, nákup, predaj, personalistika, marketing a pod. (podľa tematického zamerania tej-ktorej služby).

Príkladom spracovateľskej služby je služba „*Výber optimálnych dodávateľov*“, ktorá je schematicky znázornená na obr. 22. Táto služba má jednu spustiteľnú funkciu s názvom „*Vykonaj*“, ktorá dokáže na základe údajov dostupných v databázci a konkrétnych požiadaviek výberu zadaných používateľom vybrať optimálnych dodávateľov pre zabezpečenie požadovanej dodávky (môže ísť o viacerých dodávateľov, ak sa nedá celá dodávka zabezpečiť iba od jedného dodávateľa). O spracovateľskú službu ide preto, lebo táto služba vykonáva so vstupnými údajmi matematický výpočet, do ktorého sú zakomponované pravidlá výberu dodávateľov používané v konkrétnom podniku.

Výber optimálnych dodávateľov
<p>Funkcia: Vykonaj</p> <p>Vstupy: názvy materiálov, resp. surovín na objednanie,  <i>množstvá materiálov, resp. surovín na objednanie,</i>  požiadavky výberu a ich dôležitosť,  údaje o dodávateľoch, ich dodacích podmienkach, cenách,  kvalitatívnych parametroch a pod. (dostupné v databáze)</p> <p>Výstupy: názvy a kontaktné údaje zvolených dodávateľov,  údaje o tom, aké typy materiálov, resp. surovín a v akých  množstvách sa majú od jednotlivých dodávateľov  objednať</p>

Obr. 22: Spracovateľská služba „Výber optimálnych dodávateľov“ [Zdroj: autor]

3. **Pomocné služby** – pomocné služby sa zameriavajú na riešenie problémov technického/IT charakteru. Takéto služby môžu byť ná pomocné pri zabezpečovaní automatizácie viacerých podnikových procesov alebo podprocesov, a preto hovoríme, že majú vysokú mieru znovupoužiteľnosti naprieč podnikovými procesmi. Na rozdiel od návrhu entitných služieb a služieb majúcich pevné väzby na konkrétné podnikové procesy, je spravidla zbytočné, aby sa na ich návrhu zúčastňovali podnikoví analytici alebo experti na podnikateľskú činnosť, pretože ich význam spočíva iba v IT rovine. Ich návrh by preto mal byť úlohou len pre odborníkov z oblasti IT.

*Pomocné služby nemajú v sebe implementovanú nijakú firemnú výpočtovú logiku, ani sa v nich neodrážajú nijaké firemné postupy.* Nemajú úzku väzbu na konkrétny podnik, ale zaoberajú sa riešením určitého všeobecného IT problému. Príkladmi funkcionality, ktoré môžu pomocné služby zabezpečovať, sú najmä notifikácie o výskytte určitých udalostí technického charakteru, logovanie používateľov do systému, ošetrovanie výnimiek a chybových stavov, ktoré nesúvisia priamo s podnikovými procesmi, ale s ich IT podporou, konverzie miest, zabezpečovanie validácie dokumentov prenášaných medzi službami, replikácia údajov, šifrovanie a dešifrovanie údajov a pod. Príklad pomocnej služby je uvedený na obr. 23. Ide o službu „Import/Export tabuľiek“, ktorej

úlohou je zabezpečiť bezchybný presun údajov uložených vo forme tabuľky z jednej webovej služby do inej (t. j. transformáciu údajov uložených v tabuľke do podoby SOAP správy tak, aby túto správu mohla prijať určená služba, ale aj opačný proces – transformáciu údajov zo SOAP správy do súboru vopred definovaného formátu – napr. xls., mdb. a pod. – obsahujúceho tabuľku s údajmi).

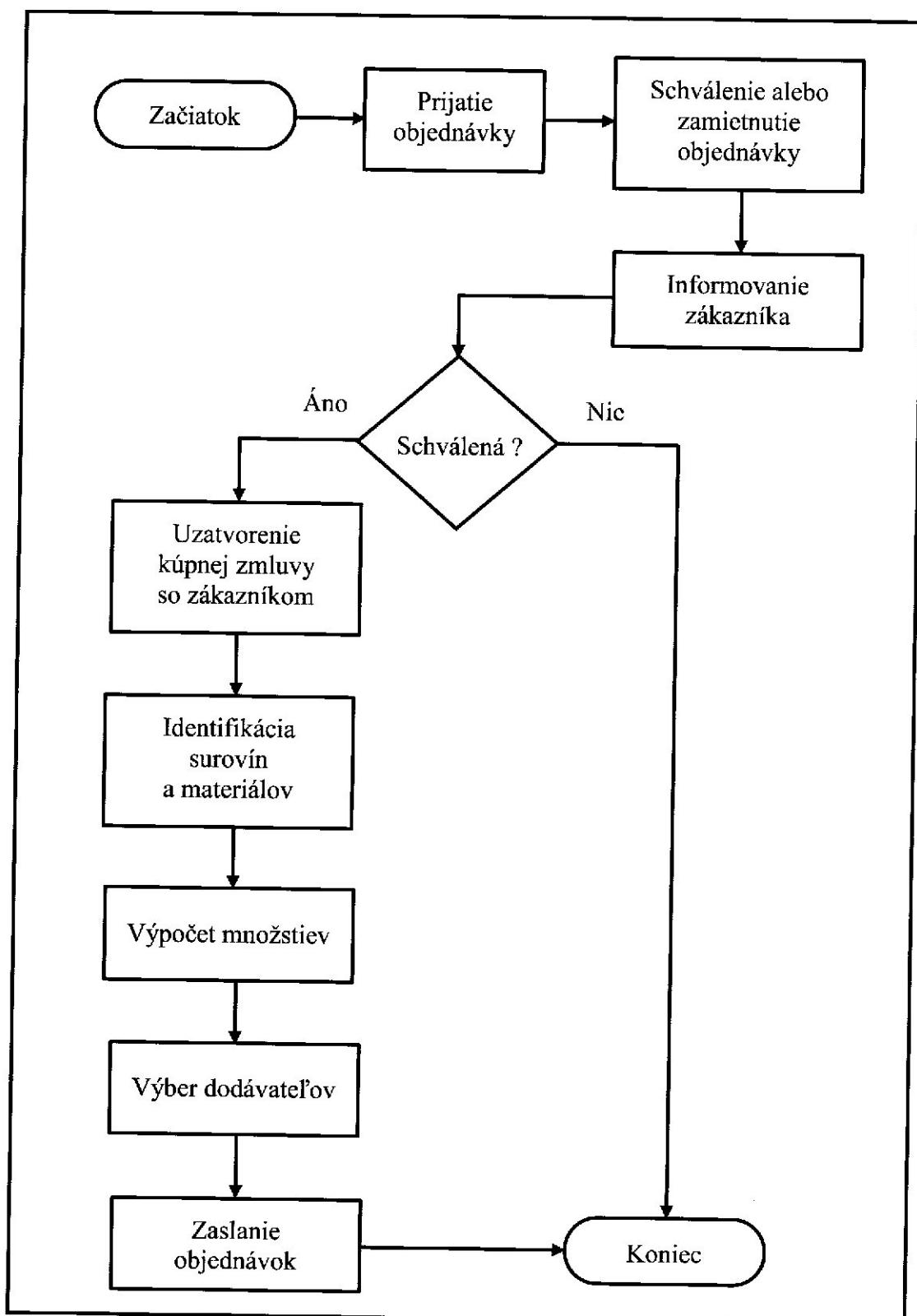
Import/Export tabuliek
<ul style="list-style-type: none"><li>• Importovať tabuľku (<u>vstup</u>: tabuľka uložená v súbore vhodného formátu; <u>výstup</u>: SOAP správa obsahujúca údaje z tabuľky)</li><li>• Exportovať tabuľku (<u>vstup</u>: údaje v SOAP správe; <u>výstup</u>: tabuľka uložená do súboru vhodného formátu obsahujúca údaje zo SOAP správy)</li></ul>

Obr. 23: Pomocná služba „Import/Export tabuliek“ a jej dve funkcie [Zdroj: autor]

### 3.4 Príklad použitia modelov služieb pri automatizácii podnikového procesu

Použitie uvedených modelov služieb predvedieme na konkrétnom príklade. V tomto príklade sa budeme zaoberať automatizáciou podnikového procesu s názvom „*Objednávanie materiálu*“, ktorý prebieha v blížšie nešpecifikovanom výrobnom podniku. Ide o pravidelne sa opakujúcu činnosť, pre ktorú si podnik stanobil jasné pravidlá jej priebehu. Je zrejmé, že ide o podproces širšieho podnikového procesu s názvom „*Nákup*“, ktorý úzko súvisí s procesom „*Výroba*“. Spúšťačom procesu „*Objednávanie materiálu*“ je zákazník, ktorý oddeleniu predaja uvažovanej výrobnej firmy zašle objednávku za účelom nákupu určitého tovaru v určitom množstve, resp. viacerých typov tovaru v určitých množstvách. Predpokladajme ďalej, že táto firma „vyrába na objednávku“. To znamená, že na sklade neudržiava zásoby svojich výrobkov a k ich výrobe dochádza až v dôsledku prijatia konkrétnej objednávky od zákazníka. V prípade, ak je objednávka schválená (t. j. nie je zamietnutá napr. z dôvodu neúplnosti údajov alebo nemožnosti príslušnú požiadavku obslúžiť), treba vykonať sériu úkonov súvisiacich so zaobstaraním všetkých typov materiálu a surovín, ktoré sú potrebné na výrobu objednaných výrobkov podľa požiadaviek zákazníka, a to v adekvátnych množstvách. Nech proces

„Objednávanie materiálu“ v uvažovanej firme pozostáva z postupnosti krokov, ktoré sú znázornené na nasledujúcom vývojovom diagrame (obr. 24).



Obr. 24: Vývojový diagram podnikového procesu „Objednávanie materiálu“  
[Zdroj: autor]

Tok podnikového procesu znázorneného na obr. 24 a jednotlivé jeho kroky môžeme vyjadriť aj slovným zápisom, a to nasledovne:

1. Prijatie objednávky od zákazníka.
2. Schválenie, resp. zamietnutie objednávky.
3. Informovanie zákazníka o schválení alebo zamietnutí objednávky.
4. Ak sme objednávku schválili (if):
  - a. Uzatvorenie kúpnej zmluvy so zákazníkom.
  - b. Identifikácia surovín a materiálov, ktoré sú potrebné na výrobu výrobkov objednaných zákazníkom.
  - c. Výpočet adekvátnych množstiev jednotlivých surovín a materiálov, ktoré treba objednať, pretože aktuálne nie sú k dispozícii na sklede.
  - d. Výber optimálnych dodávateľov na základe stanovených kritérií a ich váh (zohľadňovať sa môžu napr. cenové podmienky, dodacie podmienky, kvalitatívne parametre a pod.).
  - e. Zaslanie objednávok vybraným dodávateľom.

Zamyslime sa teraz nad tým, ktoré z týchto krovov sú vhodné na automatizáciu pomocou webových služieb a ktoré by mali byť vykonávané „manuálne“ zodpovednými pracovníkmi na príslušných oddeleniach. V praxi sme často limitovaní rozpočtom, ktorý bol na tvorbu informačného systému schválený, a môžeme operovať iba v rámci tohto rozpočtu. Z toho vyplýva, že nie vždy si môžeme dovoliť automatizovať všetko, čo sa dá automatizovať. Každá služba, resp. aplikácia navyše totiž navyšuje časovú aj finančnú náročnosť projektu. V tomto ilustratívnom príklade však budeme predpokladať, že rozpočet je dostatočne štedrý na to, aby sme si mohli dovoliť automatizovať všetko, čo sa dá.

Prvým krokom je prijatie objednávky od zákazníka. Ak predpokladáme, že sa objednávky prijímajú pomocou *webového formulára*, ktorý je na to určený, potom môžeme tento krok automatizovať pomocou webovej služby, ktorá prevezme údaje zadané do formulára a zapíše ich do databázy objednávok, v ktorej pribudne nový riadok. Táto operácia svojou povahou zodpovedá databázovej operácii CREATE, a preto by malo ísť o entitnú webovú službu určenú na manipuláciu s údajmi o objednávkach. Entitné služby sa zvyknú pomenovať rovnako ako entity, na ktoré sa vzťahujú, a preto túto službu pomenujeme „*Objednávka*“. Ak predpokladáme, že objednávky sa môžu prijímať aj inými spôsobmi (napr. telefonicky alebo emailom), môžeme opäť využiť ten istý formulár, a to tak, že ho sprístupníme zamestnancom

oddelenia predaja, zodpovedným za prijímanie objednávok, a umožníme im, aby do neho mohli sami vpisovať potrebné údaje.

Druhým krokom je schvaľovanie alebo zamietanie prijatých objednávok. Účelom tohto kroku je posúdiť, či sú k dispozícii všetky údaje, ktoré sú potrebné na to, aby sa objednávka mohla realizovať a či je vôbec realizovateľná. Môže sa totiž stať, že niekto zadá objednávku, ktorú príslušný podnik po technickej stránke nie je schopný vyrobiť, pretože nemá k dispozícii potrebné vybavenie (napr. stroje, softvér a pod.) alebo ju nestihne vyrobiť do času požadovaného zákazníkom, pretože výrobné procesy v príslušnej firme trvajú dlhšie. Posúdenie vhodnosti objednávky je pomerne zložitá úloha, ktorá si vyžaduje, aby posudzovateľ disponoval hlbšími znalosťami problematiky. Nejde teda o úlohu, ktorá by sa dala riešiť pomocou jednoduchého algoritmu, a preto ju nie je vhodné automatizovať pomocou webovej služby, resp. by pokus o takúto automatizáciu nemusel viest' k spoľahlivým výsledkom. Bude lepšie, ak posudzovanie vhodnosti objednávky, jej úplnosť a realizovateľnosť necháme na zodpovedného pracovníka. Výsledkom posudzovania objednávky bude, že v zodpovedajúcim riadku tabuľky objednávok a v stĺpci s názvom Schválené pribudne buď hodnota „Áno“ alebo hodnota „Nie“. Tento zápis môže vykonať funkcia UPDATE entitnej služby „*Objednávka*“ na základe podnetu od posudzovateľa – posudzovateľ vo svojom používateľskom rozhraní objednávku buď schváli alebo zamietne. Použitie tejto služby však neznamená automatizáciu rozhodovacej činnosti na tomto kroku, t. j. automatizáciu posudzovania objednávky. Posúdenie vykoná človek a služba iba zapíše jeho rozhodnutie do databázy.

Tretím krokom je informovanie zákazníka o výsledku schvaľovania jeho objednávky. V prípade, ak je všetko v poriadku, môžeme zákazníkovi odoslať univerzálnu správu o tom, že jeho objednávka je prijatá. Ak je, naopak, zamietnutá, potom je vhodné k tejto správe pripojiť určité vysvetlenie alebo zdôvodnenie tohto rozhodnutia. Toto zdôvodnenie by malo byť formulované posudzovateľom do určitej kolónky v jeho používateľskom rozhraní. Odoslanie tejto správy zákazníkovi je čisto technická záležitosť – nie je to úloha, ktorá by si vyžadovala určité spracovanie výrobných, ekonomických, účtovných alebo iných podnikových údajov. Na jej zabezpečenie preto môžeme využiť pomocnú službu s názvom „*Odoslať zákazníkovi správu o akceptácii objednávky*“.

Krokom č. 4 je rozhodovací blok, na ktorom si kladieme otázku, či bola objednávka schválená alebo nie. Ak nebola schválená, potom príslušná inštancia procesu „*Objednávanie materiálu*“ končí. Ak bola schválená, potom je tok inštancie procesu nasmerovaný do vetvy, ktorá pozostáva zo sekvencie piatich čiastkových krovov.

Prvým z nich je uzavorenie kúpnej zmluvy so zákazníkom. Kúpna zmluva sa po spísaní a vytlačení musí podpísať oboma zúčastnenými stranami (zákazníkom a poskytovateľom plnenia, čiže predajcom), aby bola platná. Považujme teda toto za manuálny krok, ktorý nebudeme automatizovať pomocou služieb.

Druhým čiastkovým krokom v rámci kroku č. 4 je zostavenie zoznamu všetkých typov materiálu a surovín, ktoré sú potrebné na výrobu toho, čo si zákazník objednal. Táto úloha sa dá riešiť algoritmicky a vyžaduje si spracovanie údajov o kompozícii jednotlivých výrobkov dostupných v databáze. Na jej vyriešenie preto môžeme použiť spracovateľskú službu s názvom „*Zostav zoznam surovín a materiálov*“, príp. „*Zostav zoznam komponentov*“. Výstupom tejto služby bude SOAP správa obsahujúca zoznam všetkých surovín a materiálov, ktoré treba objednať na obslúženie požiadavky zákazníka, ako aj ich potrebné množstvá, pričom tu nesmie nijaká položka chýbať a ani sa tá istá položka nesmie objaviť dvakrát (ak potrebujeme určity komponent na výrobu viacerých typov výrobkov, potom by služba mala zrátat potrebné množstvá a uviesť celkové potrebné množstvo tohto komponentu – summarizácia údajov). Táto správa bude adresovaná ďalšej (nadväzujúcej) službe, pre ktorú bude slúžiť ako vstup. Údaje o zložení výrobkov, ktoré táto služba potrebuje, získa od entitnej služby „*Výrobok*“ (od jej funkcie READ).

Tretím čiastkovým krokom v rámci kroku č. 4 je výpočet množstiev jednotlivých surovín a materiálov, ktoré treba na obslúženie požiadaviek zákazníka, ak aktuálne nie sú na skade, resp. nie sú k dispozícii v potrebnom množstve. Tento výpočet treba urobiť pre všetky položky zoznamu surovín a materiálov, ktorý je výstupom predchádzajúceho kroku (krok č. 4b). Výpočet množstiev môže zrealizovať vhodne naprogramovaná spracovateľská služba, ktorú môžeme nazwać „*Výpočet potrebných množstiev surovín a materiálov*“. Táto služba potrebuje ako vstup dostať výstup služby používanej na kroku č. 4b (môže ho dostať vo forme SOAP správy) a potrebuje tiež prístup k údajom o zložení jednotlivých výrobkov (to potrebovala aj predchádzajúca spracovateľská služba) a k údajom o aktuálnom stave zásob na skade. Keďže spracovateľské služby nezvyknú mať zabezpečený priamy prístup k databáze a tento prístup sa zvykne realizovať prostredníctvom entitných služieb, vytvoríme entitnú službu „*Výrobok*“ (na zabezpečenie prístupu k údajom o výrobkoch) a entitnú službu „*Zásoby na skade*“ (na zabezpečenie prístupu k údajom o aktuálnych skladových zásobách).

Po výpočte potrebných množstiev surovín a materiálov nasleduje výber optimálnych dodávateľov, od ktorých treba chýbajúce materiály a suroviny objednať. Pri výbere dodávateľov sa môžu brať do úvahy rozličné kritériá, najmä však rýchlosť dodávky, cena dodávky

a kvalitatívne parametre (ak sú známc). Problém výberu optimálnych dodávateľov je možné algoritmizovať s využitím vhodných viackriteriálnych optimalizačných metód operačného výskumu. Môžeme teda vytvoriť spracovateľskú službu „*Výber optimálnych dodávateľov*“, ktorá ako vstup potrebuje dostať výstup predchádzajúcej služby (t. j. vo forme SOAP správy s údajmi o množstvách surovín a materiálov, ktoré treba objednať) a taktiež potrebuje ako vstup dostať údaje o kritériach, ktoré treba pri výbere zohľadňovať, a ich dôležitosť – váhach (tieto kritériá sú bud' univerzálnie nastavené alebo ich môže zadať používateľ do používateľského rozhrania – ak ho systém na to vyzve). Vstupom musia byť tiež údaje o dodávateľoch dostupné v databáze (dajú sa zistiť pomocou entitnej služby „*Dodávateľ*“). Výstupom služby je zoznam dodávateľov, ktorým treba zaslať objednávky, a tiež údaje o tom, aké materiály a v akých množstvách treba objednať od ktorého z nich.

Posledným krokom je zaslanie objednávok vybraným dodávateľom na predtým stanovené materiály a suroviny vo vhodných množstvách. Tento krok si už nevyžaduje žiadne podnikovo orientované spracovanie údajov – ide o čisto technickú záležitosť, odoslanie vhodne zostavenej správy. Na to môžeme využiť pomocnú službu s názvom „*Odoslanie objednávok*“, ktorá ako vstup potrebuje dostať SOAP správu s výstupom predchádzajúcej služby (t. j. služby „*Výber vhodných dodávateľov*“). Údaje z tejto správy prevezme, vloží ich do vopred pripravenej šablóny pre objednávku a odošle zvolenému dodávateľovi, čo urobí pre každého dodávateľa nachádzajúceho sa na zozname.

Výstupom služby je množina objednávok odoslaných štandardizovaným spôsobom (napr. formou mailu, formou EDI – Electronic Data Interchange, cez extranet alebo v podobe SOAP správy – ak naši obchodní partneri používajú takisto informačný systém na báze SOA a používajú v ňom webovú službu na príjem objednávok). Po odoslaní všetkých objednávok príslušná inštancia procesu „*Objednávanie materiálu*“ končí.

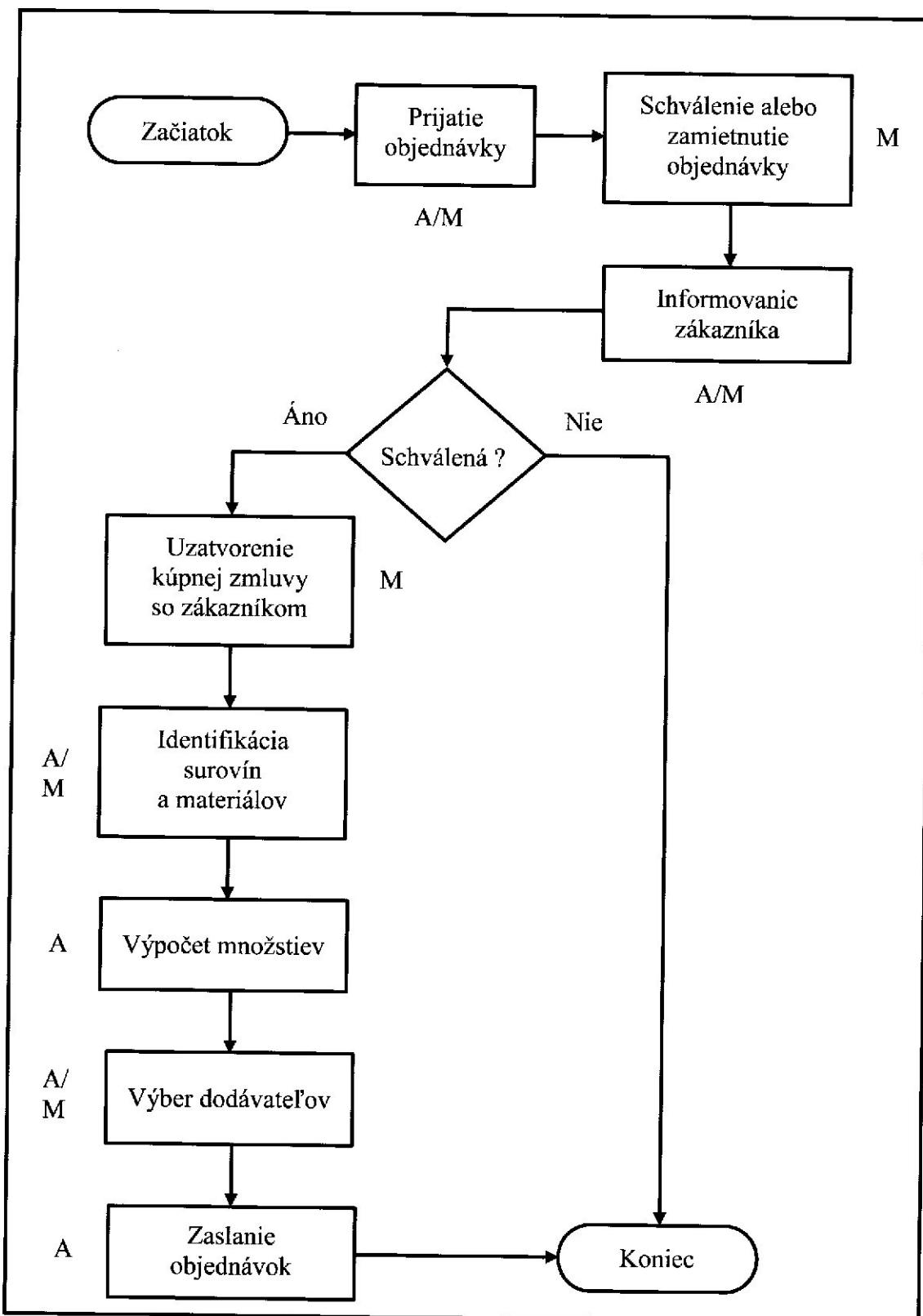
Teraz sa sústredíme na kroky č. 4b, 4c, 4d a 4e. Všetky tieto kroky na seba priamo nadvádzajú. Každá ďalšia služba v tejto postupnosti potrebuje ako vstup SOAP správu s výsledkami služby použitej na automatizáciu predchádzajúceho kroku. Vzniká tu teda reťazec štyroch služieb, ktoré dohromady zabezpečujú automatizáciu takmer celého kroku č. 4 (s výnimkou kroku č. 4a, ktorý sa má vykonávať manuálne – t. j. bez automatizácie pomocou webovej služby). V kapitole 1 sme si povedali, že služby sa dajú skladať do spolupracujúcich celkov a že sa to dá zabezpečovať buď prostredníctvom *choreografie* (t. j. bez použitia centrálneho koordinátora) alebo prostredníctvom *orchestrácie* (t. j. s použitím centrálneho koordinátora). Orchestráciu má zmysel použiť najmä vtedy, ak existujú určité špecifické pravidlá, ktoré

hovoria, kedy sa má ktorá služba spustiť. Tieto pravidlá zvyčajne zahŕňajú rozličné podmienky (if; if – else; if – elseif – else; a pod. – t. j. môžu sa v nich vyskytnúť jednoduché vetvenia neúplné, jednoduché vetvenia úplné či viacnásobné vetvenia), ale môžu zahŕňať aj cykly (for; while; do – while; t. j. cykly s vopred známym alebo s vopred neznámym počtom opakovaní, pri ktorých sa podmienka ukončenia cyklu môže vyhodnocovať pred vstupom do iterácie alebo na jej konci). Vzhľadom na to, že v tomto príklade na seba služby č. 4b, 4c, 4d a 4e nadväzujú priamo (ich závislosť je lineárna a nie je podmienená vetveniami ani cyklami), nemá zmysel zavádzat centrálneho koordinátora. Tým by sme zbytočne zvýšili zložitosť systému (stúpol by počet SOAP správ prenášaných sietou, čo by zbytočne komplikovalo odovzdávanie údajov medzi službami, a bolo by treba vytvoriť jednu službu navyše). Ako vhodnejšie kompozičné riešenie sa preto javí choreografia.

Ak do vývojového diagramu znázorňujúceho tok podnikového procesu „*Objednávanie materiálu*“ (obr. 24) doplníme vo forme komentárov informácie o tom, ktoré z jeho krokov sme sa rozhodli automatizovať pomocou webových služieb a ktoré sa majú vykonávať manuálne zodpovednými pracovníkmi bez použitia služieb, získame diagram zobrazený na obr. 25. Na tomto obrázku sú značkou „A“ označené kroky, ktorých výkon má byť plne automatizovaný pomocou určitej webovej služby. Značkou „M“ sú označené kroky, ktoré majú byť v plnej miere vykonávané manuálne, resp. bez použitia webových služieb, a značka „A/M“ označuje kroky, ktoré majú byť automatizované pomocou webovej služby, no pritom sa vyžaduje alebo priprúšťa aj zásah človeka.

Na základe analýzy predmetného podnikového procesu sme sa rozhodli pre vytvorenie nasledujúcich služieb:

- entitná služba „*Objednávka*“,
- pomocná služba „*Odoslať zákazníkovi správu o akceptácii objednávky*“,
- spracovateľská služba „*Zostav zoznam surovín a materiálov*“,
- spracovateľská služba „*Výpočet potrebných množstiev surovín a materiálov*“,
- spracovateľská služba „*Výber optimálnych dodávateľov*“,
- pomocná služba „*Odoslanie objednávok*“,
- entitná služba „*Výrobok*“,
- entitná služba „*Zásoby na sklade*“,
- entitná služba „*Dodávateľ*“.

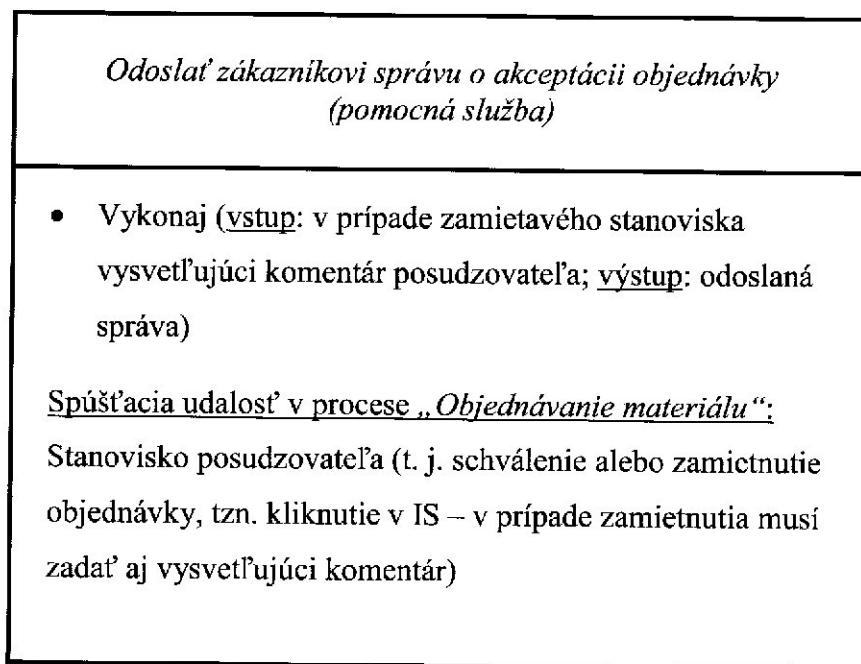


Obr. 25: Vývojový diagram podnikového procesu „Objednávanie materiálu“ rozšírený o informácie o využití webových služieb na jednotlivých jeho krokoch  
 [Zdroj: autor]

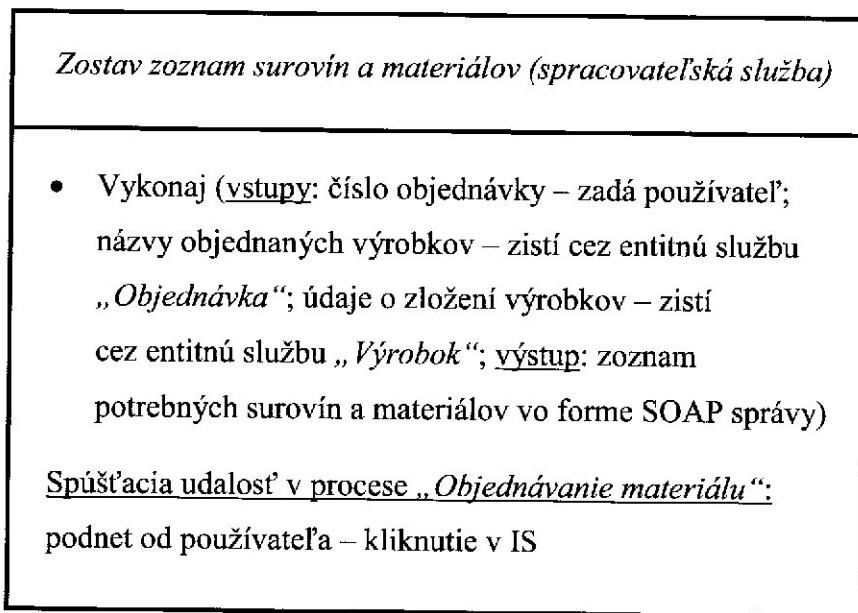
Pre všetky služby, ktoré sme sa rozhodli zaviesť do systému, vytvorime zodpovedajúce návrhy zakreslené do rámčekov. V každom rámčeku bude uvedený názov služby a v závorku za ním jej typ (t. j. použitý model služby). Ďalej budú vymenované názvy funkcií, ktoré bude táto služba zapuzdrovať (t. j. ktoré ponúka svojmu okoliu ako svoju spustiteľnú funkciu) a pre každú funkciu zoznam vstupných údajov, ktoré potrebuje, a zoznam výstupných údajov, ktoré má poskytnúť. Pre každú službu tiež uvedieme udalosť, ktorá ju má v kontexte príslušného podnikového procesu spustiť (spúšťaciu udalosť), t. j. čo má byť impulz na to, aby došlo k spusteniu služby. Spúšťacou udalosťou pre službu môže byť napr. príchod určitej správy od inej služby (t. j. jedna služba kontaktuje inú s určitou požiadavkou), podnet od používateľa informačného systému (napr. kliknutie používateľa na nejaké tlačidlo v IS), uplynutie vopred stanoveného časového intervalu či zmena určitého sledovaného údaja (napr. služba sleduje určitý údaj v databáze a keď sa tento údaj dostane do určitého intervalu, alebo presiahne určitú hranicu, je to pre službu podnet na jej spustenie). Návrhy všetkých služieb sú uvedené na obrázkoch (obr. č. 26 – 33).

<i>Objednávka (entitná služba)</i>
<ul style="list-style-type: none"><li>• Pridať_objednávku (<u>vstup</u>: údaje o novej objednávke; <u>výstup</u>: nový riadok v tabuľke objednávok)</li><li>• Poskytnúť_údaje_o_objednávke (<u>vstup</u>: špecifikácia žiadanych údajov; <u>výstup</u>: údaje žiadanej žiadateľom)</li><li>• Zmeniť_údaj_o_objednávke (<u>vstup</u>: špecifikácia toho, ktorý údaj sa má zmeniť a špecifikácia jeho novej hodnoty; <u>výstup</u>: zmenený údaj)</li><li>• Vymazať_objednávku (<u>vstup</u>: špecifikácia riadka, ktorý sa má odstrániť – alebo spoločnej charakteristiky viacerých riadkov; <u>výstup</u>: odstránený riadok alebo riadky)</li></ul> <p><u>Spúšťacia udalosť v procese „Objednávanie materiálu“:</u> príchod objednávky od zákazníka</p>

Obr. 26: Entitná služba „Objednávka“ a funkcie, ktoré sprístupňuje svojmu okoliu  
[Zdroj: autor]



Obr. 27: **Pomocná služba „Odoslať zákazníkovi správu o akceptácii objednávky“** [Zdroj: autor]



Obr. 28: **Spracovateľská služba „Zostav zoznam surovín a materiálov“** [Zdroj: autor]

*Výpočet potrebných množstiev surovín a materiálov  
(spracovateľská služba)*

- Vykonaj (vstupy: údaje v SOAP správe od služby „*Zostav zoznam surovín a materiálov*“ – t. j. zoznam potrebných surovín a materiálov; výstupy: potrebné množstvo pre každú položku zoznamu potrebných surovín a materiálov – výstup má formu SOAP správy)

Spúšťacia udalosť v procese „Objednávanie materiálu“:  
prijatie SOAP správy od služby „*Zostav zoznam surovín a materiálov*“

Obr. 29: Spracovateľská služba „Výpočet potrebných množstiev surovín a materiálov“  
[Zdroj: autor]

*Zásoby na sklade (entitná služba)*

- Pridať\_položku (vstup: údaje o novej položke; výstup: nový riadok v tabuľke zásob)
- Poskytnúť\_údaje\_o\_položke (vstup: špecifikácia požadovaných údajov; výstup: údaje žiadanej žiadateľom)
- Zmeniť\_údaj\_o\_položke (vstup: špecifikácia toho, ktorý údaj sa má zmeniť a špecifikácia jeho novej hodnoty; výstup: zmenený údaj)
- Vymazat\_položku (vstup: špecifikácia riadka, ktorý sa má odstrániť, alebo spoločnej charakteristiky viacerých riadkov; výstup: odstránený riadok alebo riadky)

Spúšťacia udalosť v procese „Objednávanie materiálu“: požiadavka od služby „*Výpočet potrebných množstiev surovín a materiálov*“ týkajúca sa aktuálneho množstva konkrétnej položky na sklade

Obr. 30: Entitná služba „Zásoby na sklade“ a jej funkcie [Zdroj: autor]

### *Výber optimálnych dodávateľov (spracovateľská služba)*

- Vykonaj (vstupy):
  - zoznam surovín a materiálov a potrebné množstvo pre každú položku tohto zoznamu – tieto údaje sú zapísané v SOAP správe od služby „*Výpočet potrebných množstiev surovín a materiálov*“;
  - požiadavky výberu a ich dôležitosť – budť ich zadá používateľ IS alebo sa zoberú implicitne nastavené požiadavky;
  - údaje o dodávateľoch, ich dodacích podmienkach, cenách, kvalita-tívnych parametroch a pod. – dostupné v databáze dodávateľov;  
výstupy: údaje o tom, aké typy surovín a materiálov sa majú v akých množstvách objednať od jednotlivých dodávateľov)

Spúšťacia udalosť v procese „Objednávanie materiálu“: príchod SOAP správy od služby „*Výpočet potrebných množstiev surovín a materiálov*“

Obr. 31: Spracovateľská služba „Výber optimálnych dodávateľov“ [Zdroj: autor]

### *Dodávateľ (entitná služba)*

- Pridať\_dodávateľa (vstup: údaje o novej položke; výstup: nový riadok v tabuľke dodávateľov)
- Poskytnúť\_údaje\_o\_dodávateľovi (vstup: špecifikácia požadovaných údajov; výstup: požadované údaje)
- Zmeniť\_údaj\_o\_dodávateľovi (vstup: špecifikácia toho, ktorý údaj sa má zmeniť, a špecifikácia jeho novej hodnoty; výstup: zmenený údaj)
- Vymazat\_dodávateľa (vstup: špecifikácia riadka/riadkov, ktorý/ktoré sa má/majú odstrániť; výstup: odstránený riadok alebo riadky)

Spúšťacia udalosť v procese „Objednávanie materiálu“: žiadosť o údaje (SOAP správa) od služby „*Výber optimálnych dodávateľov*“ alebo služby „*Odoslanie objednávok*“

Obr. 32: Entitná služba „Dodávateľ“ a jej funkcie [Zdroj: autor]

### *Odoslanie objednávok (pomocná služba)*

- Vykonaj (vstupy):

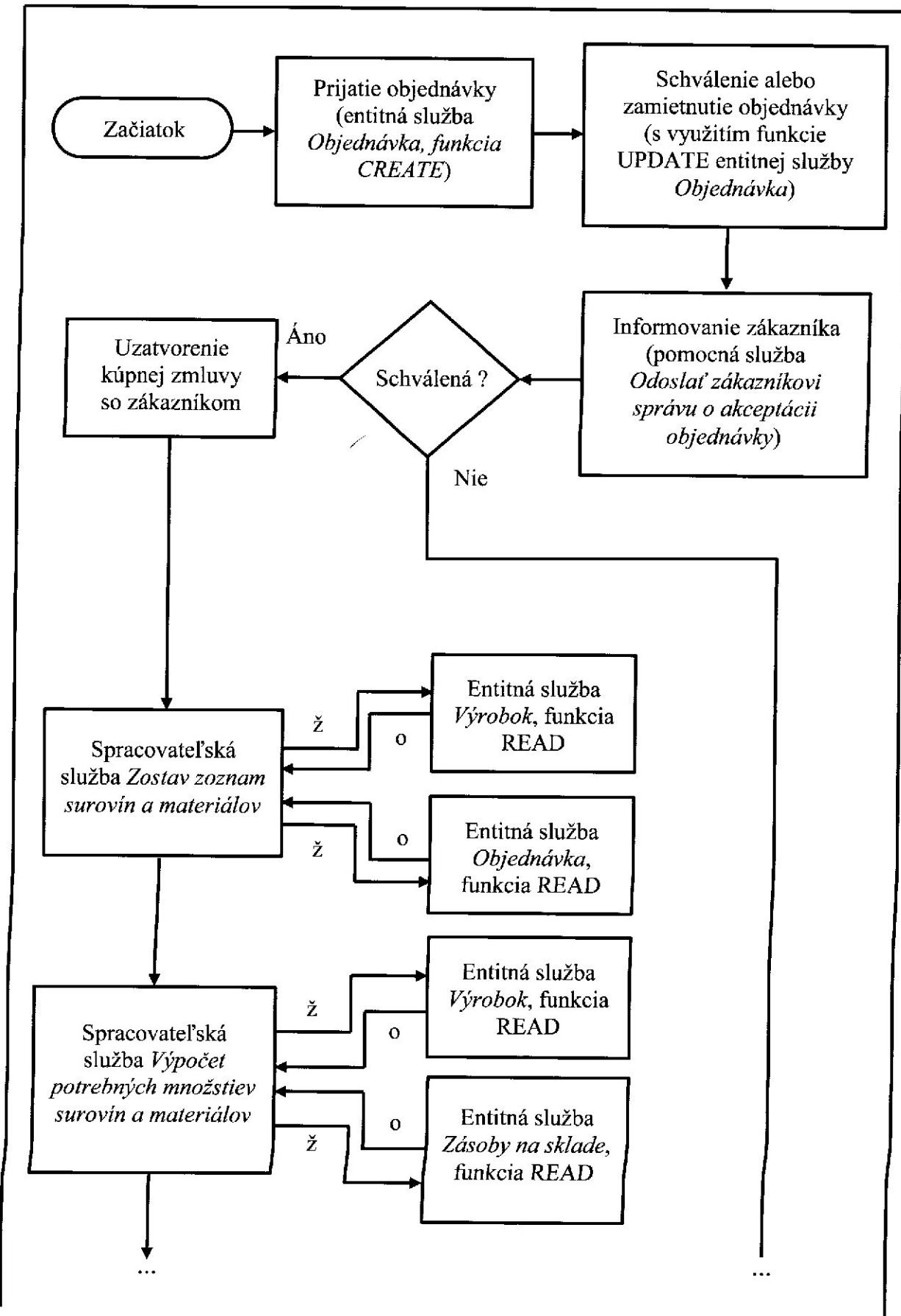
- údaje o tom, čo treba objednať od akého dodávateľa  
a v akom množstve – tieto údaje sú v SOAP správe  
od služby „*Výber optimálnych dodávateľov*“;
- kontaktné údaje o jednotlivých dodávateľoch – z entitnej  
služby „*Dodávateľ*“;  
výstupy: odoslané objednávky);

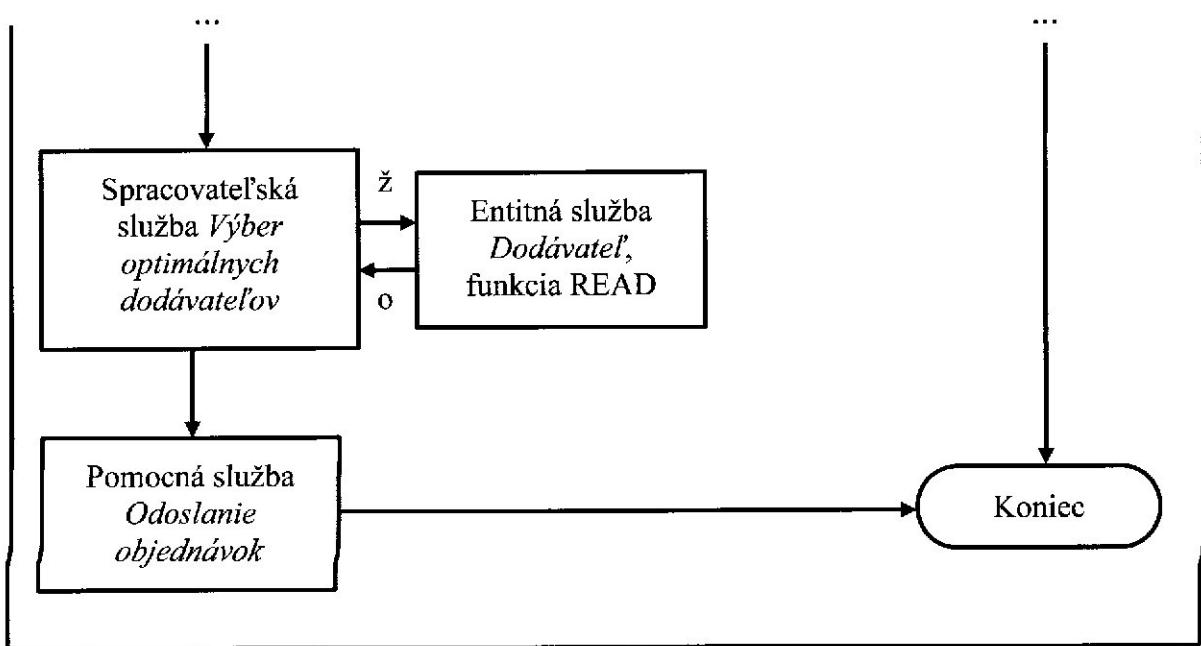
Spúšťacia udalosť v procese „*Objednávanie materiálu*“: príchod  
SOAP správy od služby „*Výber optimálnych dodávateľov*“

Obr. 33: **Pomocná služba „Odoslanie objednávok“** [Zdroj: autor]

Rámcové návrhy služieb (zobrazené na obr. 26 – 33) reprezentujú tzv. kandidátov služieb. Ide teda o predbežné návrhy, ktoré sa v priebehu implementácie systému môžu upravovať. Všimnime si, že tieto návrhy neobsahujú konkrétné zdrojové kódy (abstrahujú teda od konkrétneho programovacieho jazyka alebo iných technológií). Ide len o slovné opisy toho, aké vstupné údaje potrebujú jednotlivé funkcie tej-ktorej služby, aké výstupné údaje poskytujú a aké sú ich spúšťacie udalosti (t. j. čo sa musí stať, aby táto služba vedela, že má spustiť niektorú zo svojich funkcií). Každá funkcia, ktorá je súčasťou služby, môže mať svoju vlastnú spúšťaciu udalosť. Rámcové návrhy neobsahujú detaily o vnútorných mechanizmoch služby – t. j. neuvádzajú žiadne informácie týkajúce sa toho, akým algoritmom transformuje služba alebo jej konkrétna funkcia vstupy na výstupy. Princíp skrývania vnútorných pravidiel fungovania určitého objektu (v tomto prípade služby) pred okolitým svetom sa v informatike označuje ako *princip čiernej skrinky* a ide o často používaný princíp.

Ilustratívny príklad automatizácie podnikového procesu „*Objednávanie materiálu*“, prezentovaný v tejto podkapitole, zakončíme tým, že navrhnuté služby schematicky začleníme do vývojového diagramu, ktorý znázorňuje jeho priebeh. Vďaka tomu je vidieť očakávaný priebeh celého procesu – vidíme, ktoré jeho kroky budú vykonávané bez použitia služieb, a ktoré budú, naopak, automatizované prostredníctvom konkrétnych služieb. Vidíme tiež vzájomnú spoluprácu medzi službami. Výsledný diagram je zobrazený na obr. 34.





Obr. 34: Schéma zapojenia navrhnutých služieb do podnikového procesu „Objednávanie materiálu“ (skratka „ž“ reprezentuje „žiadosť o údaje“ a skratka „o“ zasa „odpoveď obsahujúcu požadované údaje“) [Zdroj: autor]

### Zhrnutie kapitoly 3

Webové služby sú základnými stavebnými jednotkami informačného systému vybudovaného na princípoch SOA. Rozlišujeme pritom tri základné modely, resp. typy služieb – entitné služby, služby vzťahujúce sa na podnikové procesy a pomocné služby. Entitná služba je služba, ktorá sa vzťahuje na konkrétnu podnikovú entitu (napr. zamestnanec, výrobok, oddelenie, dodávateľ atď.) a slúži na prístup k údajom o tejto entite v databáze. Ide pritom iba o štyri základné operácie s údajmi: vytvorenie nového záznamu v databáze, prečítanie záznamu, modifikovanie záznamu, vymazanie záznamu (skrátene C, R, U, D – create, read, update, delete). Ku každej podnikovej entite by sa mala vytvoriť samostatná entitná služba. Jedna podniková entita sa môže vyskytovať vo viacerých podnikových procesoch, a preto sa aj jedna entitná služba zvyčajne dá využiť na podporu automatizácie viacerých podnikových procesov. Služby vzťahujúce sa na podnikové procesy sú na rozdiel od entitných služieb obvykle úzko späté s jedným konkrétnym procesom a možnosti ich využitia na automatizáciu viacerých procesov preto bývajú veľmi nízke. Tieto služby sa delia na spracovateľské a koordinačné. Spracovateľské služby slúžia na spracovávanie údajov, t. j. na analýzu údajov, rozličné výpočty, porovnávanie, posudzovanie, vyhodnocovanie údajov a pod. Ich dôležitým znakom je, že sa vždy priamo týkajú vnútorných pravidiel, postupov a aktivít konkrétneho podniku. To

znamená, že v sebe mávajú implementovanú určitú firemnú výpočtovú logiku alebo sa v nich odrážajú určité firemné postupy, a preto ich môžeme označiť aj ako biznis služby (majú totiž úzku väzbu na konkrétny podnikový proces konkrétneho podniku a presný priebeh tohto procesu). Koordinačné služby mávajú takisto úzku väzbu na konkrétny podnikový proces. Úlohou koordinačnej služby je koordinovať (to znamená usmerňovať) činnosť iných služieb, ktoré sú jej podriadené tak, aby sa všetky tieto služby spúšťali v správnom poradí podľa pravidiel priebehu konkrétneho procesu a aby bol dosahovaný požadovaný výsledok tohto procesu. Pomocné služby sa zasa zaobrajú riešením všeobecných problémov, ktoré spadajú do oblasti IT a nemajú úzku väzbu na konkrétny podnikový proces, ba dokonca ani na podnik ako taký. Ide napr. o konverziu údajov z jedného formátu do iného, logovanie používateľov do systému, šifrovanie údajov a pod. Pomocné služby sa vďaka svojej všeobecnej povahе dajú využiť v mnohých podnikových procesoch (t. j. nie sú procesne špecifické), ba dokonca aj v mnohých podnikoch, keďže ide o riešenie všeobecných IT problémov, s ktorými sa firmy stretávajú všade vo svete.

### Literatúra ku kapitole 3

- [ERL09] ERL, T.: *SOA Servisně orientovaná architektura*. 2009. Brno: Computer Press. 672 s. ISBN 978-80-251-1886-3.
- [OAS06] OASIS: *Reference Model for Software Oriented Architectures*. 2006. [Online]. [Cit. 15. 7. 2020]. Dostupné na internete: <<http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf>>.
- [POU15] POUR, J. – GÁLA, L. – ŠEDIVÁ, Z.: *Podniková informatika*. 3. vydanie. Praha: Grada Publishing, a. s., 2015. 240 s. ISBN 978-80-247-5457-4.
- [SEN09] SENGA, E. K. – CALITZ, A. P. – GREYLING, J. H.: *An Adaptive User Interface Model Using a Service Oriented Architecture*. 2009. South African Telecommunications Network and Applications Conference [Online]. [Cit. 15. 7. 2020]. Dostupné na internete: <[https://www.researchgate.net/publication/281587126\\_An\\_Adaptive\\_User\\_Interface\\_Model\\_Using\\_a\\_Service\\_Oriented\\_Architecture](https://www.researchgate.net/publication/281587126_An_Adaptive_User_Interface_Model_Using_a_Service_Oriented_Architecture)>.

## Kapitola 4

### Životný cyklus informačného systému na báze SOA

**Životný cyklus** informačného systému vytvárajú aktivity v rámci časového úseku, ktorý sa začína úmyslom vytvoriť tento systém a končí sa vtedy, keď sa systém prestane používať [BRU12 a SEV18]. **Model životného cyklu** informačného systému predstavuje určitý zovšeobecnený prístup k riadeniu tohto cyklu. Vymedzuje, akými etapami bude prechádzať konkrétny informačný systém v rámci svojho životného cyklu a aká bude vzájomná následnosť a nadväznosť týchto etáp. Informačné systémy vybudované na báze SOA prechádzajú v rámci svojho životného cyklu týmito etapami [ERL09]:

1. *Servisne orientovaná analýza*
2. *Servisne orientovaný návrh*
3. *Vývoj služieb*
4. *Testovanie služieb*
5. *Nasadenie služieb do prevádzky*
6. *Správa služieb*

**Servisne orientovaná analýza** sa zameriava na analýzu objektu, pre ktorý navrhujeme informačný systém. Týmto objektom môže byť podnik, ale aj škola, nemocnica, banka, mestský úrad alebo iná inštitúcia či organizácia. Treba zmapovať organizačnú štruktúru tohto objektu, funkcie jednotlivých jej prvkov a procesy, ktoré v tomto objekte prebiehajú. Horizontálna štruktúra procesov sa zvykne modelovať pomocou *diagramových techník*, ktoré sú na to určené, ako napr. vývojové diagramy a BPMN diagramy. Horizontálna štruktúra procesu vyjadruje presnú postupnosť krokov, ktoré tvoria príslušný proces. V prípade potreby sa dajú na tento účel použiť aj *tabuľkové techniky*, medzi ktoré patria najmä RACI matice a rozhodovacie tabuľky. Rozhodovacie tabuľky sa hodia najmä na sprehľadnenie zložitých rozhodovacích problémov, ktoré sa môžu vyskytovať v podnikových procesoch.

Na základe diagramov znázorňujúcich jednotlivé procesy a ich kroky sa usilujeme navrhnúť služby, ktoré by sa mohli podieľať na ich čiastočnej alebo úplnej automatizácii. Výsledkom tejto fázy je zoznam tzv. *kandidátov služieb* – t. j. služieb, ktoré sa neskôr potenciálne môžu objaviť v konečnom riešení IS. Pre každú službu v tomto zozname treba určiť jej názov, typ (t. j. zvolený model služby podľa klasifikácie služieb uvedenej v kapitole 3.3), názvy a obsahovú náplň funkcii, ktoré ju majú tvoriť, a spúšťaciu udalosť, ktorá má viesť ku spusteniu konkrétnej funkcie tej-ktorej služby v kontexte konkrétneho podnikového procesu.

Pre každú funkciu treba určiť vstupné údaje, ktoré táto funkcia potrebuje, aby mohla zrealizovať očakávanú činnosť, a tiež výstupné údaje, ktoré majú byť výsledkom tejto činnosti.

**Servisne orientovaný návrh** je fázou, v ktorej treba presne vymedziť logiku zapuzdrenú v jednotlivých kandidátoch služieb. Logikou služieb rozumieme, čo má konkrétna služba, resp. konkrétna funkcia tejto služby uskutočňovať, t. j. aké konkrétnie kroky sa v nej majú vykonávať a v akom poradí. Pritom však abstrahujeme od konkrétneho programovacieho jazyka – kroky sú teda opísané iba slovne a výsledkom tohto návrhu nie je zdrojový kód. Ak má služba pozostávať z viacerých funkcií, potom musíme podrobne opísat' vnútornú logiku každej z nich. Pri servisne orientovanej analýze sme sa zaoberali iba tým, aké služby sa majú v systéme objaviť a aké funkcie majú tieto služby obsahovať (t. j. aké majú mať tieto funkcie názvy). Nezaobrali sme sa tým, aké kroky a v akom poradí sa majú objaviť v určitej funkcii konkrétnej služby. Servisne orientovaný návrh je teda rozšírením a konkretizáciou toho, k čomu sme dospeli pri servisne orientovanej analýze. Často sa pritom ukáže, že niektorí kandidáti služieb zo servisne orientovanej analýzy sa dajú zlúčiť, či naopak vôbec nie sú potrební, príp. treba vytvoriť nových kandidátov, o ktorých vytvorení sa predtým neuvažovalo. Zoznam kandidátov služieb sa preto môže meniť.

Vo fáze **vývoja služieb** dochádza k výberu konkrétneho programovacieho jazyka a vývojovej platformy, ktoré sa použijú na vytvorenie jednotlivých služieb. Často sa pritom zvyknú používať vývojové platformy .NET a J2EE. Výsledkom tejto fázy sú hotové služby naprogramované v konkrétnom jazyku, ktoré sú pripravené na použitie.

Pred definitívnym nasadením služieb do prevádzky je potrebné tieto služby dokonale **otestovať**. Testeri služieb si pritom musia klásiť napr. tieto otázky:

- Aké typy žiadateľov služieb môžu k jednotlivým službám pristupovať?
- Poskytuje služba všetko, čo sa uvádza v dokumentoch, ktoré ju opisujú?
- Akým typom chybových situácií môže byť služba vystavená?
- Ako dobre odzrkadľujú opisy služieb funkcionalitu, ktorú tieto služby poskytujú?
- Na akých kompozíciách (t. j. orchestráciách alebo choreografiách) sa jednotlivé služby majú zúčastňovať?
- Aké problémy môžu nastať pri zadávaní údajov jednotlivým službám?

Pokiaľ sa pri niektoej z týchto otázok identifikujú určité nedostatky, je potrebné ich pred nasadením služby do prevádzky odstrániť, aby sa predišlo zbytočným problémom.

Vo fáze *nasadzovania služieb do prevádzky* sa služby inštalujú na konkrétné serveri, na ktorých budú prevádzkované. Priradenie služieb k serverom sa musí riešiť tak, aby sa predchádzalo prílišnému preťažovaniu serverov, ale zároveň aby nedochádzalo ani k zbytočne nízkemu využitiu ich kapacity. Treba pritom vychádzať z očakávanej miery využívania jednotlivých služieb, ktorá sa však, samozrejme, od skutočnej miery môže odlišovať. To sa ale ukáže až počas reálnej prevádzky systému. Vo fáze *Nasadzovanie služieb do prevádzky* sa tiež riešia otázky súvisiace so správou používateľských účtov, prístupovými právami, zabezpečením prístupu k jednotlivým službám, ochranou citlivých údajov a pod.

Fáza *Správa služieb* sa zaoberá bežnými otázkami súvisiacimi so správou aplikácií, ktoré sú súčasťou informačného systému. Medzi takéto otázky patria najmä:

- Ako sledovať mieru využívania, resp. vyťaženosť jednotlivých služieb?
- Ako sledovať mieru využívania, resp. vyťaženosť jednotlivých serverov?
- Ako identifikovať úzke miesta výkonu?
- Ako do systému už nasadených služieb pridávať nové služby?
- Ako zo systému niektoré služby odoberať, alebo upravovať ich funkciaľitu podľa meniacich sa požiadaviek?
- Do akej miery jednotlivé služby vyhovujú požiadavkám ich používateľov?

## Zhrnutie kapitoly 4

Životný cyklus informačného systému začína prvotnými aktivitami, ktoré sa vykonávajú pri jeho zdrode a trvá, až kým tento systém nie je vyradený z prevádzky. Informačné systémy vybudované na princípoch SOA prechádzajú šiestimi fázami, ktorými sú servisne orientovaná analýza, servisne orientovaný návrh, vývoj služieb, testovanie služieb, nasadenie služieb do prevádzky a správa služieb. Táto publikácia sa venuje problematike tvorby informačných systémov ako prostriedkov podpory procesného riadenia podnikov, a preto je pre nás spomedzi týchto šiestich fáz najdôležitejšou hned' prvá fáza (t. j. servisne orientovaná analýza), ktorá najviac súvisí so správnym pochopením podnikových procesov a správnym počiatočným navrhnutím služieb na podporu týchto procesov (v súvislosti s procesným riadením ide o kritickú fazu). Služby, resp. ich návrhy sa dajú dodatočne upravovať aj v ďalších fázach, no vždy je najmenej prácne a časovo aj finančne nákladné, keď sa správny návrh urobí už na začiatku. Túto fazu si preto podrobnejšie rozoberieme v nasledujúcej kapitole.

## Kapitola 5

### Servisne orientovaná analýza

Prvou fázou životného cyklu IS vytvoreného podľa princípov SOA je *servisne orientovaná analýza*. Ako sme už uviedli v kapitole 4, táto fáza sa zameriava na analýzu objektu, pre ktorý navrhujeme informačný systém. Treba zmapovať organizačnú štruktúru tohto objektu, funkcie jednotlivých prvkov tejto štruktúry a procesy, ktoré v tomto objekte prebiehajú. Pri detailnejšom pohľade na jednotlivé procesy a ich kroky sa usilujeme navrhnúť služby, ktoré by sa mohli podieľať na ich čiastočnej alebo úplnej automatizácii. Výsledkom tejto fázy je vytvorenie zoznamu *kandidátov služieb* – t. j. služieb, ktoré sa neskôr potenciálne môžu objaviť v konečnom riešení IS. Hlavným zmyslom tejto fázy je teda hrubé (konceptuálne) modelovanie služieb bez detailného špecifikovania ich vnútornej logiky a funkcionality a takisto bez ohľadu na ich technologické detaile a špecifiká (programovacie jazyky, databázové systémy, operačné systémy, komunikačné systémy a ostatné technológie potrebné na neskoršiu implementáciu namodelovaných služieb).

Pre úspešné zvládnutie fázy servisne orientovanej analýzy autor predkladá nasledujúci postup (tentototo postup je formulovaný na podnikové prostredie, ale je aplikovateľný aj na iné typy objektov, ako sú napr. banky, mestské úrady, školy či nemocnice):

1. *Analýza organizačnej štruktúry podniku a funkcií jednotlivých organizačných útvarov* – poznanie organizačnej štruktúry podniku a funkcií jej jednotlivých prvkov je základným východiskom k uskutočneniu kvalitnej procesnej analýzy. Každý krok ľubovoľného podnikového procesu musí mať jasne stanoveného vykonávateľa, čiže musí byť jasné, ktorý organizačný útvar je zodpovedný za jeho vykonanie. Pritom zvyčajne ide o činnosť rutinne vykonávanú daným útvarom – t. j. funkciu tohto útvaru. Podnikové procesy teda pozostávajú z vhodne usporiadaných funkcií organizačných útvarov tak, aby toto usporiadanie viedlo k dosiahnutiu vopred stanoveného cieľa.
2. *Identifikovanie procesov a podprocesov prebiehajúcich v objekte a ich modelovanie prostredníctvom diagramových techník* – identifikujeme všetky procesy prebiehajúce v analyzovanom objekte (podniku) a v prípade potreby, ak sú nickoré procesy príliš zložité, rozčleníme ich na podprocesy, čím situáciu sprehľadníme. Medzi najvhodnejšie diagramové techniky na modelovanie podnikových procesov patria vývojové diagramy a diagramy vytvorené podľa standardu BPMN, no dajú sa použiť aj tabuľkové techniky, ako sú napr. RACI matice, ktorých výsledkom nie je diagram, ale tabuľka (viac o nich

sa dá dočítať v [JUR18]). Pritom je potrebné zakresliť priebeh všetkých identifikovaných procesov, t. j. bez ohľadu na to, či ide o *hlavné, podporné alebo riadiace procesy*. Všetky procesy sa snažíme zakresliť natoľko podrobne, aby každý nákres predstavoval jasný, zrozumiteľný a v praxi realizovateľný návod na riešenie konkrétneho typu problému a aby si vykonávateľia procesu nemuseli príliš veľa krokov alebo súvislostí domýšľať. To by viedlo k nejasnostiam, v dôsledku ktorých by jednotlivé inštancie procesu nemuseli viesť k očakávaným výsledkom.

3. *Zostavenie procesnej mapy* – po identifikovaní a zakreslení všetkých hlavných, podporných a riadiacich procesov pomocou diagramov znázorňujúcich ich priebeh zostavíme zoznam všetkých procesov a podprocesov vo forme *procesnej mapy*. Ide o diagram, ktorý vyjadruje hierarchiu procesov a sprehľadňuje procesnú štruktúru podniku. Zvyčajne sa zostavuje princípom zhora nadol, podľa ktorého najskôr v podobe obdĺžnikov zakreslíme skupiny procesov (t. j. vytvoríme skupinu pre hlavné procesy, skupinu pre riadiace a skupinu pre podporné procesy) a následne ich detailizujeme tak, že k týmto skupinám pridáme konkrétné procesy, ktoré do nich patria. Tieto procesy zakresľujeme takisto ako obdĺžniky a pripájame ich k zodpovedajúcim skupinám pomocou šípok. Ak niektorý proces pozostáva z podprocesov, zakreslíme tieto podprocesy taktiež ako obdĺžniky a pripojíme ich šípkami k tomuto procesu. Každý proces a aj podproces musí mať svoje jedinečné číslo s tým, že tieto čísla by mali odzrkadľovať hierarchiu procesov. Napr. ak skupine riadiacich procesov priradíme číslo 2 a táto skupina pozostáva z dvoch procesov, je vhodné ich označiť 2.1 a 2.2. Pritom ak proces 2.2 pozostáva z troch podprocesov, potom je vhodné ich označiť 2.2.1, 2.2.2 a 2.2.3. Ak majú byť výstupy jedného procesu vstupmi do iného procesu, je vhodné túto skutočnosť uviesť v zátvorke do príslušného obdĺžnika. Viac o procesných mapách uvádzame v kapitole 1.2.3 *Procesné mapy*.
4. *Posudzovanie jednotlivých krokov v zakreslených podnikových procesoch z hľadiska možnosti ich automatizácie* – spomedzi zakreslených podnikových procesov si zvolíme jeden proces a postupne prechádzame všetky jeho kroky jeden za druhým. Pri každom kroku sa zamyslíme nad tým, či si želáme pri jeho vykonávaní používať nejakú aplikáciu, ktorá ho bude automatizovať, alebo by mal byť tento krok vykonávaný manuálne pracovníkom, ktorý je na to určený, t. j. bez použitia aplikácií. Pritom rozlišujeme 3 situácie: daný krok by mal byť plne automatizovaný (označíme ho písmenom „A“), daný krok by mal byť automatizovaný iba čiastočne (označíme ho písmenami

„A/M“), daný krok by nemal byť automatizovaný a mal by byť vykonávaný iba manuálne (označíme ho písmenom „M“). Tento postup opakujeme pre každý zakreslený podnikový proces. Pritom však musíme podotknúť, že nie je nevyhnutné zavádzat' automatizáciu všade, kde je to možné. Zvyčajne sa to ani nedá, pretože pri projektovaní IS bývame limitovaní rozpočtom a každá ďalšia služba, ktorá sa musí navrhnuť a naprogramovať, navyšuje časovú aj finančnú náročnosť projektu. Pri určovaní toho, ktoré kroky by sa mali automatizovať a ktoré nie, by sme preto mali byť opatrní a mali by sme mať na zreteli časové a finančné obmedzenia.

5. *Stanovenie spôsobu automatizácie pre všetky kroky, ktoré majú byť automatizované* – spomedzi všetkých zakreslených podnikových procesov sa najskôr zameriame na jeden a pre každý jeho krok, ktorý má byť sčasti alebo úplne automatizovaný, musíme stanoviť spôsob, ako sa má táto automatizácia realizovať. Ak projektujeme IS na princípoch SOA, potom sa usilujeme, automatizáciu v čo najväčšej miere zabezpečiť pomocou webových služieb. Pritom máme na výber zo štyroch základných typov služieb prezentovaných v kapitole 3.3 (t. j. entitné, spracovateľské, koordinačné a pomocné). Nie je však nevyhnutné všetko automatizovať iba pomocou služieb. Môžeme tiež využiť tradičné podnikové aplikácie, ako napr. MS Excel, MS Access, Adobe Photoshop, MS Outlook, SAP a pod. Ide o praxou preverený typový aplikačný softvér<sup>4</sup> so širokými možnosťami použitia a v mnohých prípadoch by bolo nezmyselné pokúšať sa ho nahradzať webovými službami s rovnakou alebo podobnou funkcionálitou. Tento postup opakujeme pre každý podnikový proces.
6. *Vytvorenie zoznamu kandidátov služieb* – všetky služby identifikované v predchádzajúcom kroku spíšeme do jedného zoznamu. Každá služba v tomto zozname musí mať svoj názov a musí byť pri nej uvedený jej model (t. j. či ide o entitnú, spracovateľskú, pomocnú alebo koordinačnú službu). Tiež je potrebné uviesť, na automatizácii ktorých podnikových procesov sa má táto služba zúčastňovať.
7. *Rámcový návrh služieb* – pre každú službu uvedenú v zozname kandidátov služieb vytvoríme jej rámcový návrh. Tento návrh sa zakresľuje v podobe rámčeka a obsahuje:
  - názov služby;
  - typ služby (t. j. použitý model služby);

---

<sup>4</sup>Ako typový aplikačný softvér označujeme aplikácie, ktoré neboli vyrobene „na mieru“ pre konkrétnu firmu podľa jej špecifických požiadaviek, ale boli vyrobene na základe zovšeobecnených požiadaviek firiem alebo používateľov.

- zoznam funkcií, ktoré má táto služba ponúkať svojmu okoliu. Pre každú funkciu treba uviesť jej názov, zoznam vstupných údajov, ktoré potrebuje na výkon svojej činnosti, a zoznam výstupných údajov, ktoré má táto funkcia poskytnúť;
  - charakteristiku spúšťacej udalosti, t. j. udalosti, ktorá musí v kontexte konkrétneho podnikového procesu nastať, aby sa služba „dozvedela“, že má spustiť niektorú zo svojich funkcií. Spúšťacou udalosťou vo všeobecnosti môže byť napr. podnet od používateľa (kliknutie používateľa v IS), príchod SOAP správy s konkrétnym obsahom, ktorý je služba schopná rozpoznať a spracovať, či iná udalosť, ktorou môže byť príchod objednávky či iného podnetu zvonka, zmena sledovaného údaja v databáze, napr. jeho poklesnutie pod sledovanú hranicu a pod. Spúšťaciu udalosť musíme uviesť pre každý proces, na ktorého automatizáciu sa daná služba má podieľať. Každá funkcia služby by mala mať svoju vlastnú spúšťaciu udalosť. Rámcové návrhy služieb v tomto tvare sú znázormené napr. na obr. 26 – 33 (str. 103 – 107).
8. *Začlenenie navrhnutých služieb do procesných diagramov* – na záver všetky navrhované služby zo zoznamu kandidátov služieb vhodne začleníme do diagramov znázorňujúcich priebeh jednotlivých podnikových procesov. To znamená, že ak má byť konkrétny krok určitého podnikového procesu automatizovaný pomocou konkrénej služby, uvedieme názov a typ (t. j. model) služby do zátvorky k tomuto kroku. Okrem toho je potrebné zachytiť tiež spoluprácu medzi službami. Ak sa napr. niektorá spracovateľská služba obracia na entitnú službu so žiadosťou o údaje, táto entitná služba údaje vyhľadá v databáze, do ktorej má prístup, a zašle ich spracovateľskej službe, musíme túto skutočnosť zachytiť v príslušnom procesnom diagrame. Príklad začlenenia služieb do procesných diagramov uvádzame na obr. 34 (str. 108 – 109).

## Zhrnutie kapitoly 5

Fáza servisne orientovanej analýzy je úvodnou fázou pri budovaní informačného systému na báze SOA. Ide v nej pritom o analýzu objektu, pre ktorý sa tento systém navrhuje (napr. výrobný podnik, banka, vedeckovýskumná inštitúcia, inštitúcia štátnej správy a pod.). Cieľom tejto fázy je správne pochopiť procesy príslušného objektu, namodelovať ich pomocou vhodných diagramových techník a na základe toho vytvoriť prvotné návrhy služieb, z ktorých by mal budúci systém pozostávať. V tejto fáze ide len o predbežné a hrubé návrhy, ktoré sa budú detailizovať v neskorších fázach (výsledkom sú iba tzv. kandidáti služieb – čiže nie každá

z navrhovaných služieb sa musí objaviť v konečnom riešení, resp. sa nemusí objaviť presne v navrhovanom tvere). Napriek tomu je veľmi dôležité urobiť tieto návrhy čo najlepšie, pretože sa tým predíde prácnemu prerábaniu hotových služieb alebo ich návrhov v neskorších fázach a ušetrí sa množstvo finančných prostriedkov aj času. Celý postup prezentovaný v tejto kapitole je ilustrovaný na konkrétnom príklade, ktorý je uvedený v podkapitole *3.4 Príklad použitia modelov služieb pri automatizácii podnikového procesu*.

## Kapitola 6

### Najdôležitejšie otvorené štandardy v SOA

V tejto kapitole si detailne priblížíme hlavné štandardy, ktoré tvoria základ konceptu SOA. Ide najmä o tieto štandardy: *XML (Extensible Markup Language)*, *XSD (XML Schema Definition Language)*, *WSDL (Web Services Description Language)*, *UDDI (Universal Description, Discovery and Integration)* a *SOAP (Simple Object Access Protocol)*.

#### 6.1 XML

Jazyk **XML** (*Extensible Markup Language* – v preklade „Rozšíritelný značkovací jazyk“) bol vytvorený konzorciov W3C (*World Wide Web Consortium*) a bol odvodený od jazyka **SGML** (*Standard Generalized Markup Language* – v preklade „Štandardný zovšeobecnený značkovací jazyk“), ktorý vznikol už na konci 60. rokov 20. stor. Popularita jazyka XML vzrástla počas rozvoja elektronického obchodu na konci 90. rokov 20. stor., a to najmä kvôli prudkému rozvoju internetu. Význam jazyka XML spočíva najmä v jeho schopnosti priradiť údajom význam a kontext. Je to teda **jazyk slúžiaci na platformovo nezávislé uchovávanie údajov, ktoré môžu byť ľahko prenášané prostredníctvom internetových prenosových protokолов**. Postupom času boli z tohto jazyka odvodené aj ďalšie jazyky, a to **XSD (XML Schema Definition Language** – v preklade „Jazyk na definovanie XML schém“) a **XSLT (XML Transformation Language** – v preklade „Jazyk na transformáciu XML“).

XML nie je programovací jazyk, t. j. jeho účelom nie je tvorba počítačových programov. Je to jazyk na zápis, resp. reprezentáciu údajov, ktorý umožňuje priradiť jednotlivým údajom význam a zachytiť vzájomné vzťahy medzi nimi. Kód zapísaný v jazyku XML teda nevykonáva žiadnu činnosť, iba reprezentuje množinu navzájom súvisiacich údajov.

V SOA je jazyk XML veľmi dôležitý, pretože jeho pravidlá sú základom pre konštrukciu SOAP správ prenášaných medzi webovými službami, ako aj základom pre konštrukciu WSDL dokumentov opisujúcich jednotlivé webové služby. Ako uvádz Herout: „*v súčasnej dobe ide o jeden z najdôležitejších formátov výmeny dát štruktúrovaným spôsobom.*“ [HER12]. Ide o štandard, ktorého dobrými vlastnosťami sú najmä [HER12]:

- *XML opisuje údaje nezávisle na platforme, t. j. nezávisle na type počítača a operačného systému.* To umožňuje veľmi dobrú prenositeľnosť údajov, a tiež veľmi dobrú prenositeľnosť aplikácií, ktoré s nimi pracujú. Nezávislosť na platforme podporuje najmä

skutočnosť, že obsahom XML dokumentu sú textové informácie (XML je textový dokument) s tým, že informácie o kódovaní znakov sú uvedené v hlavičke dokumentu.

- *XML patrí medzi tzv. otvorené štandardy* – za jeho používanie nemusíme platiť žiadne poplatky.
- *Značky majú pevnú gramatiku.* Pri zápise značiek je treba dodržať niekoľko základných pravidiel ich zápisu s tým, že formálna správnosť zápisu sa dá ľahko skontrolovať prostredníctvom softvérových produktov na to určených, ktoré sa nazývajú *validátory*.
- *Jazyk XML je otvorený jazyk* – používateľia majú možnosť vytvárať svoje vlastné značky podľa toho, s akými údajmi potrebujú pracovať. To je rozdiel oproti uzavretým jazykom, medzi ktoré patrí napr. jazyk *HTML (Hypertext Markup Language* – v preklade „*Hypertextový značkovací jazyk*“) Tento jazyk pozostáva z množiny konkrétnych značiek s konkrétnym významom, spomedzi ktorých si používateľ môže vybrať, no nemá možnosť vytvárať vlastné značky.
- *Značkovanie v jazyku XML umožňuje nielen uložiť údaje, ale aj opisať, resp. vysvetliť ich význam a vzájomné vzťahy* – to je ďalší z rozdielov oproti jazyku HTML – zatiaľ čo HTML sa zameriava na to, akým spôsobom sa údaje zobrazia na obrazovke, cieľom XML je uchovať údaje, zachytiť ich význam a vzájomné vzťahy. K údajom sa tak prispájajú dodatočné údaje, tzv. metadáta, ktoré ich bližšie opisujú.

Nevýhodou uchovávania údajov v XML súboroch je to, že značky, ktoré údaje ohraňujú a pridávajú im význam, zmnohonásobujú veľkosť týchto súborov. Z tohto dôvodu sa XML nehodí na uchovávanie veľkých objemov údajov (rádovo stovky MB) [HER12].

Jedným zo základných pojmov terminológie XML je **element**. Je to označenie pre začiatočnú a koncovú značku vrátane údaja uvedeného medzi nimi, napr. <Zákazník> Jozef Trnka </Zákazník>. V každom XML dokumente sa musí nachádzať práve jeden *koreňový element*, t. j. element, ktorý „obaľuje“ všetky ostatné elementy nachádzajúce sa v tomto dokumente.

Značky môžu obsahovať písmená (s diakritikou aj bez), číslice a niektoré špeciálne znaky, akými sú pomlčka („-“), bodka („.“) a podčiarkovník („\_“). Názov značky sa nikdy nesmie začínať číslicou a musí ísť o súvislý reťazec znakov (t. j. nesmie obsahovať medzeru). Okrem toho sa v názvoch značiek rozlišujú veľké a malé písmená, tzn. značka <Zákazník> je iná značka než značka <zákazník>.

Každá začiatočná značka musí mať aj svoju uzaváraciu značku, ktorá má rovnaký názov, no je odlišená znakom „/“. Napr. začiatočná značka <Zákazník> musí mať koncovú značku </Zákazník>. Element, ktorý neobsahuje žiadne údaje medzi svojimi značkami, sa nazýva *prázdný element*. Prázdný element je ekvivalentom hodnoty NULL, ktorá je známa z databáz. Táto hodnota nie je rovná nule, ale vyjadruje, že určitý údaj je neznámy, resp. nie je k dispozícii.

Elementy sa do seba vnárajú, čím vzniká hierarchická štruktúra pozostávajúca z podadených a nadadených elementov. Elementy, ktoré sú vo vnútri určitého iného elementu, sa označujú ako *vnorené*. Element, ktorý ich obsahuje, resp. zapuzdruje, sa zasa označuje ako *rodičovský*. Koreňový element je teda rodičovský element, ktorý obsahuje všetky ostatné elementy určitého XML dokumentu. Vnorené elementy sa pre lepšiu prehľadnosť dokumentu zvyknú odsadzovať tabulátorom s tým, že vzniknuté medzery sú zvyčajne pri spracovaní dokumentu ignorované.

Príkladom jednoduchého XML dokumentu je nasledujúci dokument, ktorý je nositeľom údajov o dvoch zákazníkoch:

```
<?xml version="1.0" encoding="UTF-8"?>
<Zákazníci>
    <Zákazník>
        <ID_zákazníka>0035</ID_zákazníka>
        <Meno>Jozef</Meno>
        <Priezvisko>Trnka</Priezvisko>
        <Adresa>
            <Ulica>Furdekova 15</Ulica>
            <Mesto>Bratislava</Mesto>
            <PSČ>852 35</PSČ>
            <Štát>Slovensko</Štát>
        </Adresa>
        <Telefónne_číslo>0903332434</Telefónne_číslo>
    </Zákazník>
    <Zákazník>
        <ID_zákazníka>0036</ID_zákazníka>
        <Meno>Peter</Meno>
```

```

<Priezisko>Rýchly</Priezisko>
<Adresa>
    <Ulica>Šustekova 25</Ulica>
    <Mesto>Bratislava</Mesto>
    <PSČ>852 35</PSČ>
    <Štát>Slovensko</Štát>
</Adresa>
<Telefónne_číslo>0904111222</Telefónne_číslo>
</Zákazník>
</Zákazníci>

```

Údaje, ktoré sú zapísané v tomto XML dokumente, by mohli byť zapísané aj v tabuľke *Zákazník* pozostávajúcej zo zodpovedajúcich riadkov a stĺpcov. V tom prípade by táto tabuľka mala tento tvar:

Tab. 6: Tabuľka Zákazník [Zdroj: autor]

ID	Meno	Priezv.	Ulica	Mesto	PSČ	Štát	Tel_č.
0035	Jozef	Trnka	Furdekova 15	BA	852 35	SR	0903332434
0036	Peter	Rýchly	Šustekova 25	BA	852 35	SR	0904111222

Vidíme teda, že jazyk XML je len jeden z spôsobov pre zápis údajov. Je však špecifický niektorými výhodami, ktoré sme už spomínali.

Súčasťou elementov môžu byť aj *atribúty*. Atribút musí mať svoj názov a môže mu priradená konkrétna hodnota pomocou symbolu „=“. Je to teda dvojica *názov\_atribútu* = "hodnota" (napr. *ročník* = "3"). Atribúty sa pritom môžu vyskytovať len vnútri začiatočných značiek elementov – v ich koncových značkách sa vyskytnúť nesmú. V nasledujúcom príklade vidíme, ako môžeme pomocou atribútu *jednotka* bližšie špecifikovať význam údaja „0,7“, ktorý je súčasťou elementu *hmotnosť\_tovaru*:

```
<hmotnosť_tovaru jednotka ="kg"> 0,7 </hmotnosť_tovaru>
```

Ak má byť znak úvodzoviek súčasťou hodnoty atribútu, je treba celú hodnotu dať do apostrofov. Tvorcovia XML dokumentov sa musia často rozhodnúť, či pre zápis metadát (t. j. údajov, ktoré umožňujú bližšie špecifikovať význam nejakého konkrétneho údaja zapísaného v XML dokumente) použijú atribúty alebo vnorené elementy. Ako uvádzá Herout: „*Atribúty majú tú výhodu, že predstavujú kompaktnejší zápis. Navyše platí, že ich je často možné ľahšie (elegantnejšie) spracovať podpornými technológiami. Na druhej strane majú atribúty tú nevýhodu, že na ich poradí nezáleží, a teda sa toto poradie nedá nijako skontrolovať. Ďalšou nevýhodou je, že sa nemôžu v jednej značke opakovat.*“ [HER12].

Ak by sme metadáta o hodnote „0,7“ z predchádzajúceho príkladu chceli zapísat výlučne pomocou elementov (t. j. bez využitia atribútov), môžeme to urobiť takto:

```
<hmotnosť_tovaru>
    <hodnota>0,7</hodnota>
    <jednotka>kg</jednotka>
</hmotnosť_tovaru>
```

Na uchovávanie dlhších textových reťazcov (napr. viet) sa obvykle nepoužívajú atribúty, ale sa tieto reťazce vkladajú medzi značky so zodpovedajúcim označením opisujúcim ich obsah.

XML súbory môžu byť **kódované** buď pomocou *národných znakových sád* (napr. ISO-8859-2, Windows-1250) alebo pomocou *univerzálnych znakových sád* (UTF-8, UTF-16). Implicitne (t. j. ak nie je uvedené inak) bývajú kódované v znakovej sade UTF-8, prípadne UTF-16. **Znaková sada** je súbor znakov, ktoré sa môžu objaviť v textovom dokumente kódovanom podľa tejto sady s tým, že každý znak v tejto sade má priradený jedinečný kód, ktorý ho odlišuje od ostatných znakov. Kódovanie dokumentu slúži na to, aby mohli byť jednoznačne rozpoznané všetky znaky (t. j. písmená, číslice, interpunkčné znamienka, ale aj tzv. biele znaky ako napr. medzera, tabulátor, koniec riadka a pod.) a aby nemohlo dôjsť k zámene znakov, resp. ich chybnej identifikácii. Ak sa použije nesprávne kódovanie, môžu sa stratiť niektoré špecifické písmená, akými sú napr. Ľ, Š, Č, Ž, ľ, š, č, ž. Ako sme už povedali, znakové sady sa členia na *národné* a *univerzálné*. Každá znaková sada je charakteristická tým, že každý jeden znak, ktorý táto sada rozpoznáva, resp. podporuje, v nej má priradený jedinečný identifikačný kód, vďaka ktorému nemôže dôjsť k zámene znakov, a tým k nesprávnej interpretácii textu, a to pri použití ľubovoľného operačného systému. Pre počítačovú reprezentáciu slovenských

znakov s diakritikou sa vyvinulo viacero *národných znakových sád*, napr. ISO-8859-2 a Windows-1250. Sada ISO-8859-2 však neobsahuje znak „€“, a preto už v súčasnosti nie je možné túto sadu plnohodnotne používať [STU07].

Na rozdiel od národných znakových sád *univerzálne znakové sady* (označované ako Unicode) zahŕňajú znaky všetkých jazykov používaných kdekoľvek na svete (t. j. vrátane Grécka, Ruska, Číny, Japonska a pod.), ide preto o veľmi rozsiahle znakové sady, ktoré umožňujú rozpoznať a preniesť enormné množstvo znakov. Ako uvádzia Centrum výpočtovej techniky STU v Bratislave na svojej webovej stránke: „*Konkrétnu reprezentáciu znakov v univerzálnom kódovaní Unicode v textovom súbore určuje tzv. Unicode Transformation Format (UTF). Napr. pri použíti formátu UTF-16 sú pre uloženie každého znaku potrebné 2 bajty – t. j. veľkosť textového súboru by bola dvojnásobná oproti súboru v národnej kódovej stránke. Podstatne efektívnejší je preto formát UTF-8 štandardizovaný ako ISO/IEC 10646, ktorý pre všetky znaky obsiahnuté v znakovej sade ASCII (t. j. číslice, písmená bez diakritiky, atď.) vyžaduje iba 1 bajt a pre reprezentáciu slovenských znakov s diakritikou 2 bajty.*“ [STU07].

Kódovanie znakov v XML súbore sa dá nastaviť prostredníctvom elementu *xml* a atribútu *encoding*, ktorý sa zvykne dávať vždy na začiatok XML dokumentu, napr.:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Atribút *version* označuje verziu XML jazyka. Štandardne sa používa verzia 1.0, no existuje aj verzia 1.1. Rozdiel medzi nimi je však len minimálny a týka sa predovšetkým použitia niektorých špeciálnych znakov v Unicode.

Každý XML dokument by mal prejsť kontrolou validity (t. j. *validáciou*), pri ktorej sa zistuje, či je tento dokument *správne štruktúrovaný* (z angl. *well-formed*) a sú v ňom dodržané všetky pravidlá jazyka XML. Podstatné meno *validácia* znamená „overovanie platnosti“ a prídavné meno *validný* znamená „platný“. Pri kontrole sa sleduje najmä týchto **11 pravidiel**:

1. Musí existovať práve jeden koreňový element (element *xml* označujúci verziu jazyka XML a použité kódovanie sa však zvykne písat pred koreňový element).
2. Každá začiatočná značka musí mať k nej prináležiacu ukončovaciu značku.
3. V začiatočných a ukončovacích značkách elementov sa musia presne zhodovať veľké a malé písmená, pretože XML patrí medzi tzv. *case-sensitive jazyky*.

4. Elementy sa nesmú navzájom prekrižovať, t. j. ak určitý rodičovský element obsahuje v sebe vnorený element, potom sa musí najskôr ukončiť vnorený element a až potom rodičovský element – vnorený element musí byť vnorený ako celok.
5. Názvy elementov a atribútov musia byť súvislé reťazce (t. j. bez medzery).
6. Názvy elementov a atribútov sa musia začínať písmenom – nesmú sa začínať číslicou, interpunkčným znamienkom či bielym znakom (medzery, tabulátory a pod.). Číslice a interpunkčné znamienka sa však môžu nachádzať za prvým znakom názvu elementu alebo atribútu.
7. Hodnoty atribútov musia byť z oboch strán uzavorené v úvodzovkách alebo apostrofoch (apostrofy sa zvyknú používať v prípadoch, ak úvodzovky majú byť súčasťou reťazca tvoriaceho hodnotu atribútu, no dajú sa použiť aj v prípade, ak sa úvodzovky v tomto reťazci nenachádzajú).
8. V jednom elemente sa nemôžu nachádzať dva alebo viaceré atribúty s rovnakým názvom.
9. Atribúty sa smú vyskytovať len v začiatočnej značke elementu.
10. Komentáre nemôžu byť navzájom vnorené a nemôžu byť ani vo vnútri značiek (komentár sa vkladá v tvare: <!-- komentár -->). To, že sa nemôžu vnárať, znamená, že ak sme začali komentár, musíme ho najskôr ukončiť a až potom môžeme začať ďalší komentár.
11. Znakové údaje nesmú obsahovať znaky „<“ a „&“ – to sa týka nielen údajov uvedených medzi začiatočnou a koncovou značkou určitého elementu, ale aj názvov značiek ako takých, názvov atribútov a hodnôt atribútov. Znak „>“ je vo všetkých týchto prípadoch povolený.

Kontrola správnej štruktúrovanej XML dokumentu sa dá uskutočniť napr. prostredníctvom štandardných webových prehliadačov, akými sú napr. Google Chrome alebo Mozilla Firefox. Tie však môžu mať problém s akceptáciou niektorých národných znakov, a to aj napriek tomu, že v XML dokumente je správne nastavené kódovanie. Býva to spôsobené najmä tým, že je vo webovom prehliadači nastavená iná znaková sada než tá, v ktorej je kódovaný XML dokument. Lepšie je preto využiť špecializované nástroje na validáciu XML dokumentov, z ktorých mnohé sú voľne dostupné on-line. Takýmto nástrojom je napr. *XML Validator* na webovej stránke organizácie W3C, ktorá jazyk XML vyvinula: [https://www.w3schools.com/xml/xml\\_validator.asp](https://www.w3schools.com/xml/xml_validator.asp). Aplikácia schopná skontrolovať validitu XML dokumentu sa označuje ako **XML parser**.

## 6.2 XSD

*Hierarchická štruktúra XML dokumentu sa dá presne definovať prostredníctvom XML schémy zapísanej prostredníctvom jazyka XSD (XML Schema Definition Language – v preklade „Jazyk na definovanie XML schém“). XML dokument preto predstavuje konkrétnu inštanciu k nemu prislúchajúcej schémy. Inými slovami, XML schéma zapísaná prostredníctvom jazyka XSD predstavuje určitý predpis, resp. šablónu na vytvorenie XML dokumentu a tento dokument je implementáciou tejto šablóny nesúcou konkrétnu údaje podľa štruktúry a dátových typov stanovených v schéme.* V XML schémach bývajú definované pravidlá a obmedzenia, ktoré musí XML dokument spĺňať, aby mohol byť považovaný za *platný*, t. j. *validný* vzhladom na konkrétnu XML schému.

*Jazyk XSD umožňuje definovať dátové typy elementov a atribútov, ktoré sa majú objavíť v určitom XML súbore, ako aj poradie elementov v tomto súbore.* Tvorcovia XSD súborov majú možnosť si zvoliť konkrétnu dátovú typu z množstva **základných typov**, no majú možnosť vytvárať aj **vlastné (používateľsky deklarované) dátové typy**. Používateľsky deklarovaný dátový typ môže byť buď **jednoduchý** alebo **zložený**. Deklarácia jednoduchého dátového typu sa začína značkou *simpleType* a vzniká odvodením od niektorého zo základných dátových typov prostredníctvom reštrikcie, ktorá sa k tomuto typu doplní. Deklarácia zloženého typu sa začína značkou *complexType* a je to typ pre taký element XML súboru, ktorý vo svojom vnútri obsahuje jeden alebo viac iných elementov, alebo má vo svojej začiatoknej značke jeden alebo viac atribútov.

Každá XML schéma (v podobe textového súboru s príponou .xsd alebo ako streamovaný dátový prúd) musí byť zapísaná v súlade s pravidlami tvorby XML dokumentov. To znamená, že musí prejsť validáciou, či ide o správne štruktúrovaný (*well-formed*) XML dokument. Začína sa preto hlavičkou definujúcou verziu XML a spôsob kódovania znakov:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Koreňový element XML schémy má vždy názov *schema* a pomocou atribútu *xmlns* sa v ňom zvyknú definovať **názvy menných priestorov**, používaných v schéme, a ich **prefixy** (napr. *xs*). Prefix je skratkou, ktorá nahradza názov menného priestoru a pre jednoduchosť sa zvykne používať vo zvyšku kódu namiesto názvu tohto priestoru. *Vďaka menným priestorom vieme v prípade potreby prepojiť viacero samostatných XML schém (t. j. .xsd súborov) a vytvoriť z nich jeden súvislý celok.* To môže byť výhodné najmä vtedy, ak sa na tvorbe jednej

schémy podieľa viaceri rôznych tvorcov a výsledná schéma má vzniknúť spojením ich kódov. Pri obyčajnom nakopírovaní všetkých kódov do jedného súboru by sa totiž mohlo stať, že sa v tomto súbore objavia definície elementov s rovnakým názvom, no odlišným významom alebo dátovým typom, čo sa nesmie stať. Jedna schéma totiž nemôže obsahovať dva elementy s rovnakým názvom, ale odlišným dátovým typom – to sa smie vyskytnúť iba vtedy, ak patria do odlišných menných priestorov.

XML schémy teda môžu, ale nemusia byť rozdelené na viaceri menných priestorov. Existuje však jeden špecifický menný priestor, ktorý by mala obsahovať každá schéma. Tento priestor má názov v tvare nasledujúcej URL adresy: "http://www.w3.org/2001/XMLSchema". Ide o základný menný priestor, v ktorom sa nachádzajú definície všetkých kľúčových slov jazyka XSD, ako sú *element*, *attribute*, *string*, *integer*, *complexType*, *restriction* a pod. Pre tento menný priestor sa zvykne používať prefix *xs*, aj keď je prípustné zaviesť pre neho namiesto tohto prefixu akýkoľvek iný prefix. Tento menný priestor a jeho prefix sa zvyknú *specifikovať* v koreňovom elemente XML schémy (t. j. v elemente s názvom *schema*) pomocou atribútu *xmlns* (skratka z angl. *XML namespace*), a to takto:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">  
</xs:schema>
```

*Každý menný priestor musí mať svoj jedinečný názov* (unikátny reťazec znakov, ktorým sa odlišuje od názvov ostatných menných priestorov používaných v tej istej schéme). *Prefixy menných priestorov* nachádzajúce sa v tej istej XML schéme *musia byť taktiež unikátne* – ak by sme dvom alebo viacerým rôznym menným priestorom priradili ten istý prefix, patril by tento prefix iba tomu mennému priestoru, ku ktorému sme ho priradili naposledy.

Ak chceme v XML schéme používať viaceré menné priestory, treba ich názvy a prefixy *specifikovať* v koreňovom elemente *schema*. Ak si napr. želáme do schémy zaviesť menné priestory „Peter“ a „Karol“ a odkazovať sa na ne prefixami *p* a *k*, zapíše sa to takto:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"  
    xmlns:p="Peter"  
    xmlns:k="Karol">  
</xs:schema>
```

V súčasnosti je zvykom zaviesť do každej XML schémy tzv. *cieľový menný priestor* (z angl. *target namespace*), aj keď to nie je nevyhnutné a XML schéma môže byť validná aj bez neho. *Cieľový menný priestor združuje všetky menné priestory nachádzajúce sa v jednej XML schéme a vďaka nemu sa vieme v iných XML alebo XSD dokumentoch odkázať na tuto schému ako na celok.* Ak ho chceme do XML schémy zaviesť, môžeme to vykonat pomocou atribútu *targetNamespace* v koreňovom elemente *schema*.

Vo vyššie uvedenom príklade sa pri mennom priestore s názvom "Peter" predpokladá, že ide o menný priestor, ktorý je v samostatnom .xsd súbore označený práve ako *targetNamespace* a vďaka nemu sme schopní pripojiť celý tento súbor a všetko, čo je v ňom definované, k nami vytváranej schéme. Pri mennom priestore "Karol" sa rovnako predpokladá, že ide o cieľový menný priestor samostatného .xsd súboru.

Cieľový menný priestor by mal mať, takisto ako ostatné menné priestory, svoj názov a prefix. V každej XML schéme sa však môže vyskytovať jeden tzv. **východiskový menný priestor**, ktorý nemá stanovený žiadny prefix a vzťahuje sa na všetky elementy, atribúty a dátové typy, definované v schéme, pri ktorých nie je uvedený žiadny prefix menného priestoru. *Je výhodné, označiť ako východiskový menný priestor práve cieľový menný priestor,* vďaka čomu nemusíme pri definovaní elementov, atribútov a dátových typov uvádzať prefix cieľového menného priestoru.

Ak chceme do XML schémy zaviesť cieľový menný priestor s prefixom (napr. *t* ako *targetNamespace*), môžeme to vykonat takto:

```
<xs:schema targetNamespace="Naša_schéma"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:t="Naša_schéma"
    xmlns:p="Peter"
    xmlns:k="Karol">
</xs:schema>
```

Ak chceme do XML schémy zaviesť cieľový menný priestor bez prefixu, t. j. označiť ho ako východiskový menný priestor, zapíšeme to takto:

```
<xs:schema targetNamespace="Naša_schéma"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

```

    xmlns="Naša_schéma"
    xmlns:p="Peter"
    xmlns:k="Karol">
</xs:schema>
```

Ukážme si, ako pomocou cielového menného priestoru prepojiť dve samostatné XML schémy – t. j. ako sa v jednej XML schéme odkázať na inú XML schému. Majme schému predpisujúcu štruktúru údajov o zákazníkoch, ktoré chceme uchovávať (schéma č. 1), a schému osobitne predpisujúcu štruktúru údajov o adrese zákazníkov (schéma č. 2). Uvedieme najskôr kód schémy č. 2:

```

<xs:schema targetNamespace="Adresa_zákazníka"
            xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns="Adresa_zákazníka">
<xs:complexType name="AdresaType">
    <xs:sequence>
        <xs:element name="Ulica" type="xs:string"/>
        <xs:element name="Mesto" type="xs:string"/>
        <xs:element name="PSČ" type="xs:string"/>
        <xs:element name="Štát" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
</xs:schema>
```

Kód schémy č. 1 potom môže byť nasledovný:

```

<xs:schema targetNamespace="Náš_zákazník"
            xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns="Náš_zákazník"
            xmlns:a="Adresa_zákazníka">
<xs:import schemaLocation="Adresa_zákazníka.xsd"
            namespace="Adresa_zákazníka"/>
<xs:element name="Zákazník">
    <xs:complexType>
```

```

<xs:sequence>
    <xs:element name="Meno" type="xs:string"/>
    <xs:element name="Priezvisko" type="xs:string"/>
    <xs:element name="Telefónne_číslo" type="xs:token"/>
    <xs:element name="Adresa" type="a:AdresaType"/>
</xs:sequence>
</xs:complexType>
</xs:clement>
</xs:schema>

```

XML dokument, ktorý je validný vzhľadom na túto kombináciu dvoch XML schém, je napr.:

```

<z:Zákazník schemaLocation="Náš_zákazník_zákazník.xsd"
    xmlns:z="Náš_zákazník"
    xmlns:a="Adresa_zákazníka">
    <z:Meno>Peter</z:Meno>
    <z:Priezvisko>Veselý</z:Priezvisko>
    <z:Telefónne_číslo>0903 89 91 11</z:Telefónne_číslo>
    <z:Adresa>
        <a:Ulica>Bulíkova 15</a:Ulica>
        <a:Mesto>Bratislava-Petržalka</a:Mesto>
        <a:PSČ>852 35</a:PSČ>
        <a:Štát>Slovensko</a:Štát>
    </z:Adresa>
</z:Zákazník>

```

Ako sme už uviedli, zmyslom XML schém je stanovovať názvy elementov, ktoré sa majú objaviť v XML dokumentoch od nich odvodených, ich poradie a dátové typy, ako aj názvy a dátové typy ich atribútov. Nasledovným XSD kódom definujeme element *Názov\_tovaru* dátového typu *string*:

```
<xs:element name="Názov_tovaru" type="xs:string"/>
```

Tento zápis definuje element s názvom *Názov\_tovaru*, v ktorom má byť uložený textový reťazec. Znak „/“ na konci začiatocnej značky je zároveň zakončením príslušného elementu XML schémy, a preto už nie je potrebné k tomuto elementu pridať aj jeho koncovú značku. Prefix *p* označuje, že tento element patrí do určitého konkrétneho menného priestoru, ktorý má takýto prefix.

V XML schémach sa pri definovaní elementov (a tiež atribútov) dajú použiť **základné dátové typy** spadajúce do nasledujúcich kategórií [HER12]:

- Dátumy a časy** – do tejto kategórie patria dátové typy uvedené v tab. 7. Aj keď sa to môže zdať zvláštne, pomlčky a nuly na začiatku hodnôt uvedených v tabuľke je potrebné pri napĺňaní XML dokumentu konkrétnymi údajmi reálne použiť (všimnite si napríklad, že pri dátovom type *gDay*, ktorý vyjadruje poradové číslo dňa v mesiaci, sa na začiatku musia za sebou nachádzať tri pomlčky, za ktorými nasleduje číslo dňa). Pri nedodržaní presnej formy zápisu by hodnoty v XML dokumente neboli validné (t. j. platné) vzhľadom na dátové typy uvedené v XML schéme.

Tab. 7: Základné dátumové a časové typy v XSD [HER12]

Dátový typ	Význam	Spôsob zápisu	Priklad
date	dátum	YYYY-MM-DD	2010-10-30
time	čas	HH:MM:SS	11:34:55
gYear	rok	YYYY	0001 až 9999
gMonth	mesiac	--MM	--01 až --12
gDay	deň	--DD	--01 až --31
gYearMonth	rok a mesiac	YYYY-MM	2010-10
gMonthDay	mesiac a deň	--MM-DD	--12-25

- Logický dátový typ** – do tejto kategórie patrí len jediný dátový typ, ktorým je *boolean*. Prvok XML dokumentu spadajúci do tohto typu môže nadobúdať len jednu z dvoch možných hodnôt, a to 1 alebo 0. Prípadne sa môžu použiť ekvivalentné hodnoty *true* (pravda) a *false* (nepravda).
- Reálne čísla** – patria sem dva dátové typy, ktorými sú *float* a *double*. Reálne čísla sú také čísla, ktoré môžu (ale nemusia) obsahovať desatinnú čiarku. *Float* predstavuje

4-bajtové číslo a *double* je 8-bajtové číslo, t. j. každé jedno číslo zaberá v pamäti pamäťové miesto s veľkosťou príslušného počtu bajtov.

4. **Desiatkové čísla** – do tejto kategórie patrí typ *decimal*, ktorý sa dá použiť pre celé aj reálne čísla zapísané prostredníctvom desiatkových číslic. Typ *decimal* má však len malé možnosti reštrikcií (v súvislosti s vytváraním používateľsky deklarovaných jednoduchých dátových typov), a preto sa pre celé čísla obvykle používajú iné dátové typy s väčšími možnosťami reštrikcií.
5. **Celé čísla** – do tejto kategórie patrí množstvo dátových typov, ktoré sa členia do dvoch kategórií – **dátové typy definované na zľava aj sprava uzavretom intervale a dátové typy definované na zľava alebo sprava otvorenom intervale**. Medzi *dátové typy definované na zľava alebo sprava otvorenom intervale* patria:
  - a. *integer* – kladné a záporné čísla vrátane nuly,
  - b. *nonPositiveInteger* – záporné čísla a nula,
  - c. *negativeInteger* – záporné čísla,
  - d. *nonNegativeInteger* – kladné čísla a nula,
  - e. *positiveInteger* – kladné čísla.

*Dátové typy definované na zľava aj sprava uzavretom intervale* sa členia na dva podtypy: **znamienkové** a **neznamienkové**. Znamienkové typy sú také, ktoré pripúšťajú kladné aj záporné čísla a nulu, neznamienkové sú zasa také, ktoré pripúšťajú iba kladné čísla a nulu.

Znamienkové aj neznamienkové typy a ich dolné a horné hranice sú uvedené v tab. 8.

Tab. 8: Základné intervalové znamienkové a neznamienkové dátové typy v XSD [HER12]

Dátový typ	Veľkosť	Dolná hranica intervalu	Horná hranica intervalu
<i>Long</i>	8 bajtov	-9 223 372 036 854 775 808	9 223 372 036 854 775 807
<i>Int</i>	4 bajty	-2 147 483 648	2 147 483 647
<i>Short</i>	2 bajty	-32 768	32 767
<i>Byte</i>	1 bajt	-128	127

Tab. 8: druhá časť

Dátový typ	Veľkosť	Dolná hranica intervalu	Horná hranica intervalu
<i>unsignedLong</i>	8 bajtov	0	18 446 744 073 709 551 615
<i>unsignedInt</i>	4 bajty	0	4 294 967 295
<i>unsignedShort</i>	2 bajty	0	65 535
<i>unsignedByte</i>	1 bajt	0	255

**6. reťazce** – v XML súboroch je možné použiť 4 základné dátové typy pre textové reťazce, ktoré sa navzájom odlišujú prípustným počtom bielych znakov, ktoré sa môžu nachádzať na začiatku, vo vnútri alebo na konci reťazca. Medzi tieto typy patria:

- a. *string* – dátový typ pre ľubovoľný reťazec s ľubovoľným počtom medzier, tabulátorov a koncov riadkov. Tento dátový typ teda zohľadňuje rozdelenie textu na riadky, ako aj odsadzovanie riadkov o ľubovoľný počet bielych znakov od ľavého okraja.
- b. *normalizedString* – dátový typ pre reťazec, ktorý takisto môže obsahovať ľubovoľný počet znakov, no XML procesor (program spracovávajúci XML dokument) pri jeho spracovaní odstráni konce riadkov, tabulátory a znaky pre návrat vozíka na tlačiarne<sup>5</sup>. Počet medzier v reťazci nie je obmedzený, t. j. môžu sa vyskytovať na začiatku, vo vnútri aj na konci riadka a môžu utvárať súvislé zoskupenia. Najväčším obmedzením tohto dátového typu oproti predchádzajúcemu je skutočnosť, že celý reťazec musí byť tvorený len jedným riadkom.
- c. *token* – dátový typ pre reťazec, ktorý môže obsahovať ľubovoľný počet znakov, no XML procesor pri jeho spracovaní odstráni konce riadkov, znaky pre návrat vozíka tlačiarne, tabulátory, *medzery na začiatku reťazca a medzery na jeho konci a takisto zoskupenia medzier nachádzajúce sa v ľubovoľnej časti reťazca* (t. j. v nijakej časti reťazca nezostanú dve alebo viaceré medzery ihneď za sebou – všetky takéto zoskupenia medzier sa zredukujú iba na jednu medzeru).

<sup>5</sup> Z angl. CR, t. j. carriage return – ide o netlačiteľný znak, ktorého úlohou je resetovať pozíciu tlačiarenského zariadenia tak, aby sa toto zariadenie nastavilo na začiatok konkrétneho riadka textu. To má využitie napr. pri prechode na nový riadok alebo pre účely zvýraznenia textu – tučné písmo, podčiarknuté písmo a pod.

- d. *ID* – dátový typ určený na tvorbu jedinečných (t. j. neopakujúcich sa) identifikátorov. Takéto identifikátory zvyknú tvoriť napr. hodnoty primárnych kľúčov v databázach, no majú aj iné využitie (napr. jedinečné skratky miest v rámci štátu – BA, SI, KE a pod.). Každý rovnako nazvaný element tohto dátového typu preto musí v XML dokumente, odvodenom od zodpovedajúcej schémy, obsahovať unikátnu hodnotu. Zároveň platí, že nesmie obsahovať biele znaky a musí sa začínať písmenom.
7. **URI** – do tejto kategórie patrí iba jeden dátový typ, ktorý má názov *anyURI*. Skratka URI znamená *Uniform Resource Identifier* (t. j. „jednotný identifikátor zdrojov“). Tento typ má veľmi podobnú funkciu ako atribút *href* v jazyku HTML. To znamená, že slúži na zápis odkazov na webové stránky (URL adresy, z angl. *Unified Resource Locator*, t. j. „jednotný lokalizátor zdrojov“), odkazov na konkrétné súbory v adresárovej štruktúre určitého servera alebo iného počítača a umožňuje aj prácu s ďalšími internetovými službami, ako napr. Telnet, FTP, Gopher a pod. Element dátového typu *anyURI* môže byť v XML dokumente, odvodenom od zodpovedajúcej schémy, zapisaný napr. takto:

```
<webstránka>https://www.youtube.com/</webstránka>
<súbor>D:/Škola/Hospodárska_informatika/prednáška1.ppt</súbor>
```

**Jednoduché používateľsky deklarované dátové typy (*simpleType*) sa vytvárajú doplnením reštrikcií (t. j. obmedzení) k základným dátovým typom.** To znamená, že ak k niektorému zo základných dátových typov doplníme obmedzenie, ktoré zúži jeho pôvodný rozsah, vzniká nový dátový typ, ktorý musí dostať určitý názov, a takto vytvorený dátový typ označujeme ako jednoduchý používateľsky deklarovaný dátový typ.

Pri tvorbe jednoduchých používateľsky deklarovaných dátových typov môžeme v XML schémach použiť nasledujúce **typy reštrikcií**:

- *Reštrikcia číselného rozsahu* – reštrikcie tohto typu je možné aplikovať na číselné, dátumové a časové dátové typy, pričom môže ísť o tieto konkrétné reštrikcie:
  - *minInclusive* – stanovenie dolnej hranice intervalu prípustných hodnôt zahŕňajúceho aj stanovenú hodnotu (zľava uzavretý interval),

- *minExclusive* – stanovenie dolnej hranice intervalu prípustných hodnôt nezahrňajúceho stanovenú hodnotu (zľava otvorený interval),
- *maxInclusive* – stanovenie hornej hranice intervalu prípustných hodnôt zahŕňajúceho aj stanovenú hodnotu (sprava uzavretý interval),
- *maxExclusive* – stanovenie hornej hranice intervalu prípustných hodnôt nezahrňajúceho stanovenú hodnotu (sprava otvorený interval).

Platí, že ak je v reštrikcii uvedená len dolná hranica prípustného intervalu, potom sa za hornú hranicu nového dátového typu automaticky považuje horná hranica pôvodného základného dátového typu. Naopak, ak je stanovená iba horná hranica, potom sa za dolnú hranicu nového dátového typu považuje dolná hranica pôvodného základného dátového typu. Reštrikcie číselného rozsahu je možné kombinovať tak, aby bola novému dátovému typu stanovená naraz dolná i horná hranica. Nasledujúci úryvok zdrojového kódu je príkladom aplikovania takýchto reštrikcií na celočíselný dátový typ *nonNegativeInteger*:

```
<xs:simpleType name="PocetStudentovType">
  <xs:restriction base="xs:nonNegativeInteger">
    <xs:minInclusive value="1"/>
    <xs:maxExclusive value="20"/>
  </xs:restriction>
</xs:simpleType>
```

- *Reštrikcia počtu platných čísl* – táto reštrikcia určuje, z akého maximálneho počtu číslík smú pozostávať čísla zodpovedajúce konkrétnemu dátovému typu. Nasledujúci kód v jazyku XSD ukazuje, ako sa dá takúto reštrikciu pripojiť k dátovému typu *nonNegativeInteger*. V tomto kóde vytvárame jednoduchý používateľsky deklarovaný dátový typ s názvom *MnožstvoTovaruNaSkladeType*, a to tak, že k základnému typu *nonNegativeInteger* pripájame obmedzenie, podľa ktorého môžu hodnoty elementov novovytvoreného typu pozostávať maximálne z piatich číslí (t. j. prípustné sú len celé čísla z intervalu od 0 do 99 999 z oboch strán uzavoreného; číslo 100 000 by už totiž obsahovalo 6 číslí a bolo by v rozpore s reštrikciou):

```
<xs:simpleType name="MnožstvoTovaruNaSkladeType">
```

```

<xs:restriction base="xs:nonNegativeInteger">
    <xs:totalDigits value="5"/>
</xs:restriction>
</xs:simpleType>

```

- *Reštrikcia počtu platných desatinných miest v číslе* – táto reštrikcia sa dá aplikovať len na dátový typ *decimal*. Udáva maximálny počet číslic, ktoré smie obsahovať desatinná časť čísla (t. j. jeho časť za desatinou bodkou). Táto reštrikcia sa zvykne kombinovať s reštrikciou celkového počtu číslic. Vďaka tomu vieme presne stanoviť, z akého maximálneho počtu číslic môže pozostávať číslo tohto typu, a maximálne kolko z týchto číslic sa môže nachádzať za desatinou bodkou (t. j. v desatinnej časti čísla). Nasledujúci príklad ukazuje kombináciu týchto dvoch reštrikcií, ku ktorým sa pripája aj reštrikcia stanovujúca dolnú hranicu intervalu prípustných hodnôt. Ide teda o skíbenie troch reštrikcií. Číslo zodpovedajúce jednoduchému používateľsky deklarovanému typu *CenaTovaruType* musí byť vyššie alebo prinajmenšom rovné nule, smie pozostávať maximálne z piatich číslic a z toho maximálne dve sa smú nachádzať za desatinou bodkou. Napr. číslo 512.32 splňa všetky tieto kritériá, no číslo 3.147 nie (za desatinou bodkou sa nachádza privlča číslic, hoci reštrikcia pre celkový počet číslic nie je porušená). Na to, aby číslo mohlo byť považované za platné, totiž musia byť splnené všetky reštrikcie.

```

<xs:simpleType name="CenaTovaruType">
    <xs:restriction base="xs:decimal">
        <xs:totalDigits value="5"/>
        <xs:fractionDigits value="2"/>
        <xs:minInclusive value="0"/>
    </xs:restriction>
</xs:simpleType>

```

- *Reštrikcia počtu znakov* – tento typ reštrikcie sa dá aplikovať na všetky dátové typy určené na zápis znakových retázcov. Existujú tri spôsoby, ako môžeme pomocou tej vymedziť počet znakov v reťazci:

- *presné stanovenie počtu znakov reťazca pomocou elementu length* – reťazec musí pozostávať z vopred presne stanoveného počtu znakov, medzi ktoré sa započítavajú aj biele znaky,
- *stanovenie minimálnej dĺžky reťazca pomocou elementu minLength* – určenie minimálneho prípustného počtu znakov v reťazci (vrátane bielych znakov),
- *stanovenie maximálnej dĺžky reťazca pomocou elementu maxLength* – určenie maximálneho prípustného počtu znakov v reťazci (vrátane bielych znakov).

Nasledujúci príklad ukazuje, ako túto reštrikciu pripojiť k dátovému typu *string* a vytvoriť tak nový jednoduchý používateľsky deklarovaný dátový typ:

```
<xs:simpleType name="PoužívateľskéMenoType">
  <xs:restriction base="xs:string">
    <xs:minLength value="8"/>
    <xs:maxLength value="20"/>
  </xs:restriction>
</xs:simpleType>
```

- *Presné vymenovanie (t. j. enumerácia) hodnôt* – tento typ reštrikcie sa dá aplikovať na všetky základné dátové typy s výnimkou logického typu *boolean*. Vďaka enumeračnému dátovému typu vieme presne vymenovať všetky prípustné hodnoty, ktoré môže nadobúdať určitý element v XML súbore (v každom časovom okamihu smie nadobúdať len jednu z týchto hodnôt). Pokial' by nadobúdal hodnotu, ktorá nezodpovedá ani jednej z vymenovaných hodnôt, išlo by o porušenie reštrikcie a XML dokument obsahujúci takýto element by neboli validný vzhľadom na príslušnú XML schému. [HER12]. Použitie enumerácie demonštruje nasledujúci úryvok kódu v jazyku XSD.

```
<xs:simpleType name="FarbyType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="červená"/>
    <xs:enumeration value="čierna"/>
    <xs:enumeration value="zelená"/>
    <xs:enumeration value="modrá"/>
```

```

<xs:enumeration value="biela"/>
<xs:enumeration value="tyrkysová"/>
</xs:restriction>
</xs:simpleType>

```

- *Aplikovanie vzorov pomocou regulárnych výrazov* – tento typ reštrikcie sa dá aplikovať na všetky základné dátové typy okrem logického typu *boolean*, no najčastejšie sa používa v spojitosti s typmi pre zápis znakových reťazcov. Pomocou elementu *pattern* a atribútu *value* sa stanoví tzv. *maska* (označovaná aj ako „vzor“ z angl. pattern) zapísaná prostredníctvom *regulárnych výrazov*. Tieto výrazy sa nazývajú regulárnymi, pretože existujú presné pravidlá ich zápisu. Maska stanovuje rozličné podmienky, ktoré musia byť dodržané v znakovom reťazci. Tieto podmienky sa týkajú napr. počtu veľkých a malých písmen, ich poradia, resp. presnej pozície v reťazci, ale aj počtu a umiestnenia interpunkčných znamienok a bielych znakov. *Regulárne výrazy* sa zapisujú takto [HER12]:
  - \d{3} – reťazec musí obsahovať práve tri číslice zapisané v desiatkovej sústave,
  - \d{1,3} – jedna, dve alebo tri číslice zapisané v desiatkovej sústave,
  - \p{L} – jedno akékoľvek písmeno,
  - \p{Lu} – jedno akékoľvek veľké písmeno,
  - \p{Ll} – jedno akékoľvek malé písmeno,
  - \p{p} – interpunkcia (oddeľovače),
  - \s – biele znaky (medzera, tabulátor, koniec riadka).

Na jednotlivé regulárne výrazy je možné použiť *opakovače*, ktoré sa uvádzajú až za výraz (t. j. nie pred neho) – toto pravidlo je dôležité nielen z hľadiska správneho zápisu masiek, ale aj z hľadiska ich správneho čítania. Existujú tri typy opakovačov so špecifickým významom:

- ? – nikdy alebo len raz,
- \* – nikdy, jedenkrát alebo viackrát,
- + – jedenkrát alebo viackrát.

Regulárne výrazy sa vo vnútri masky dajú spájať do rozličných kombinácií. Užitočnými kombináciami sú napr. [HER12]:

- " $\backslash p\{Lu\}\backslash p\{Ll\}+ \backslash p\{Lu\}\backslash p\{Ll\}+$ " – meno a priezvisko (napr. „Peter Trnka“),
- " $\backslash p\{Lu\}\backslash p\{Ll\}+ \backslash d+$ " – ulica (napr. „Gorkého 3“),
- " $\backslash d\{3\} \backslash d\{2\}$ " – poštové smerové číslo (napr. „852 35“),
- " $\backslash p\{Lu\}\backslash p\{Ll\}+[ \backslash p\{L\}\backslash p\{Ll\}]^* \backslash d^*$ " – mesto, resp. mestská časť (napr. „Bratislava“, „Praha 10“, „Ústí nad Orlicí 1“).

Na príklade nasledujúcich dvoch XML schém demonštrujeme dva spôsoby, ako vyššie uvedené reštrikcie využiť pri definovaní určitého elementu.

### Schéma č. 1:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="PSČ">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:pattern value="\d\{3\} \d\{2\}"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
</xs:schema>
```

### Schéma č. 2:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="PSČ" type="PSČType"/>
  <xs:simpleType name="PSČType">
    <xs:restriction base="xs:string">
      <xs:pattern value="\d\{3\} \d\{2\}"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

*Poznámka k schéme č. 2:* nezáleží na tom, či najskôr zdefinujeme dátový typ *PSČType* a až potom ho použijeme pri definovaní elementu *PSČ*, alebo ho najskôr použijeme a až potom zdefinujeme. Obidve riešenia sú akceptovateľné a nie je medzi nimi žiadny rozdiel – t. j. ak je určitý XML dokument validný pre jedno riešenie schémy, potom je validný aj pre druhé.

Podľa schémy č. 1 aj schémy č. 2 (obidve jej potenciálne riešenia) je validný napr. tento jednoduchý XML dokument:

```
<PSČ>852 35</PSČ>
```

Po oboznámení sa s pravidlami tvorby jednoduchých používateľsky deklarovaných dátových typov môžeme pristúpiť k pravidlám pre tvorbu zložených dátových typov. Pripomíname, že za **zložený používateľsky deklarovaný dátový typ** považujeme typ pre element XML dokumentu odvodeného od zodpovedajúcej schémy, ak pre element XML dokumentu platí jedna z týchto skutočností:

- obsahuje jeden alebo viacero vnorených elementov,
- obsahuje jeden alebo viacero atribútov,
- obsahuje jeden alebo viacero vnorených elementov a súčasne jeden alebo viacero atribútov.

Zamerajme sa najskôr na situáciu, keď je zložený dátový typ určený pre element XML dokumentu, ktorý obsahuje jeden alebo viacero vnorených elementov. Aj takýto dátový typ sa v XML schémach deklaruje pomocou elementu *element* s atribútom *name* špecifikujúcim názov nového dátového typu. Do neho však musí byť vnorený element *complexType*, ktorý hovorí, že ide o zložený používateľsky deklarovaný dátový typ. Vo vnútri elementu *complexType* sa zasa musí nachádzať aspoň jeden z trojice elementov *sequence*, *choice* a *all* alebo ich ľubovoľná kombinácia. Vo vnútri týchto elementov sa špecifikujú názvy a dátové typy elementov, ktoré majú byť vnorené do rodičovského elementu. Tieto vnorené elementy pritom môžu byť buď základných dátových typov alebo používateľsky deklarovaných dátových typov (jednoduchých či zložených). Význam výrazov *sequence*, *choice* a *all* v jazyku XSD je nasledovný:

- *sequence* – vnorené elementy sa vo vnútri rodičovského elementu v XML dokumente odvodenom od zodpovedajúcej XML schémy smú vyskytovať len vo vopred stanovenom poradí, ktoré sa musí presne dodržať,
- *choice* – vymenúva, ktoré vnorené elementy sa potenciálne môžu objaviť vo vnútri rodičovského elementu v XML dokumente odvodenom od zodpovedajúcej XML schémy, no spomedzi týchto elementov sa v jeho vnútri smie objaviť vždy iba jeden,

- *all* – špecifikuje, ktoré vnorené elementy sa s určitosťou musia objavíť vo vnútri rodičovského elementu v XML dokumente odvodenom od zodpovedajúcej XML schémy, pričom na poradí ich výskytu nezáleží.

Príkladom tohto pomerne zložitého zápisu súvisiaceho s definovaním zložených dátových typov je kód nasledujúcej XML schémy. Všimnime si, že koreňovým elementom každého XML dokumentu odvodeného od tejto schémy má byť element s názvom *Zákazník*.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="Zákazník">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Meno" type="xs:string"/>
      <xs:element name="Priezvisko" type="xs:string"/>
      <xs:element name="Adresa">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Ulica" type="xs:string"/>
            <xs:element name="Mesto" type="xs:string"/>
            <xs:element name="PSČ" type="xs:string"/>
            <xs:element name="Štát" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Telefónne_číslo" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Príkladom XML dokumentu vyhovujúcemu vyššie uvedenej XML schéme je nasledujúci dokument.

```

<?xml version="1.0" encoding="UTF-8"?>
<Zákazník>
    <Meno>Peter</Meno>
    <Priezvisko>Rýchly</Priezvisko>
    <Adresa>
        <Ulica>Šustekova 25</Ulica>
        <Mesto>Bratislava</Mesto>
        <PSČ>852 35</PSČ>
        <Štát>Slovensko</Štát>
    </Adresa>
    <Telefónne_číslo>0904111222</Telefónne_číslo>
</Zákazník>

```

V predchádzajúcim príklade sa v každom XML dokumente odvodenom od uvedenej XML schémy smie element *Zákazník* vyskytnúť iba raz. To znamená, že každý takýto dokument by smel obsahovať údaje vždy iba o jednom zákazníkovi. V XML schémach však máme možnosť ovplyvniť prípustný počet výskytov určitého elementu v XML dokumentoch, ktoré sú ich inštanciami, a to pomocou atribútov *minOccurs* (určuje minimálny nevyhnutný počet výskytov elementu na určitom mieste XML dokumentu) a *maxOccurs* (určuje maximálny prípustný počet výskytov elementu na určitom mieste XML dokumentu, ktorý nesmie byť prekročený).

Ak nastavíme *minOccurs="0"* a *maxOccurs= "unbounded"*, vyjadríme tým, že určity element sa v XML dokumente smie nachádzať ľubovoľný počet krát, t. j. počet jeho výskytov nie je nijako limitovaný (nemusí sa v ňom nachádzať vôbec, alebo sa môže nachádzať ľubovoľný počet krát).

Ak atribúty *minOccurs* a *maxOccurs* umiestnime priamo do niektorého z elementov *sequence*, *choice* alebo *all*, môžeme pomocou nich špecifikovať jednotné pravidlá prípustného počtu výskytov platiace pre všetky elementy, ktoré sú do nich vnorené. T. j. ak položíme napr. *minOccurs="0"* a *maxOccurs="1"*, vyjadríme tým, že každý z elementov vnorených do elementu *sequence* (príp. *choice* alebo *all*) sa v zodpovedajúcim XML dokumente odvodenom od príslušnej XML schémy smie vyskytnúť bud' vôbec alebo iba raz. Naopak, ak atribúty *minOccurs* a *maxOccurs* umiestnime do definície konkrétneho elementu, potom sa budú ich hodnoty vzťahovať len na tento element.

Elementy *sequence*, *choice* a *all* sa môžu do seba vnárať, čím môžeme vytvárať ich rozličné kombinácie podľa aktuálnych potrieb. Príkladom takého vnárania je nasledujúci kód XML schémy určenej pre dokumenty nesúce údaje o tovaroch uvedených na objednávke priatej od zákazníka. V tomto príklade predpokladáme, že sa na jednej objednávke môže vyskytnúť ľubovoľný počet rozličných druhov tovarov a o každom z nich chceme evidovať štyri rozličné údaje – identifikačné číslo, názov tovaru, objednané množstvo a jednotkovú cenu. XML schéma pre takýto dokument má tento tvar:

```
<?xml version="1.0" encoding="Windows-1250"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="TovaryNaObjednávke">
<xs:complexType>
<xs:choice minOccurs="1" maxOccurs="unbounded">
<xs:element name="Tovar">
<xs:complexType>
<xs:sequence>
<xs:element name="ID_tovaru" type="xs:ID"/>
<xs:element name="Názov_tovaru" type="xs:string"/>
<xs:element name="Množstvo_tovaru" type="xs:decimal"/>
<xs:element name="Jednotková_cena" type="xs:decimal"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:schema>
```

Prikladom XML inštancie, ktorá je validná vzhľadom na túto schému, je dokument s týmto kódom:

```
<?xml version="1.0" encoding="Windows-1250"?>
<TovaryNaObjednávke>
<Tovar>
```

```

<ID_tovaru>R023</ID_tovaru>
<Názov_tovaru>Jogurt Rajo malinový</Názov_tovaru>
<Množstvo_tovaru>40</Množstvo_tovaru>
<Jednotková_cena>0.41</Jednotková_cena>
</Tovar>
<Tovar>
<ID_tovaru>O129</ID_tovaru>
<Názov_tovaru>Čokoláda Orion biela</Názov_tovaru>
<Množstvo_tovaru>35</Množstvo_tovaru>
<Jednotková_cena>0.77</Jednotková_cena>
</Tovar>
</TovaryNaObjednávke>

```

Teraz sa zamerajme na špecifikovanie *dátových typov pre atribúty*. Ako už vieme, atribúty sú v XML dokumentoch vždy súčasťou nejakého konkrétneho elementu a bývajú uvedené v jeho začiatočnej značke. Atribút smie byť len základného dátového typu alebo jednoduchého používateľsky deklarovaného dátového typu (t. j. atribúty nikdy nesmú byť zloženého používateľsky deklarovaného dátového typu).

Na definovanie atribútu sa v jazyku XSD používa element *attribute* s atribútom *name*, ktorý určuje názov definovaného atribútu, ďalej s atribútom *type*, ktorý určuje jeho dátový typ, ako aj s atribútom *use*, ktorý určuje, či je výskyt definovaného atribútu povinný. Ak nastavíme *use=„optional“*, potom je výskyt tohto atribútu nepovinný, no ak nastavíme *use=„required“*, potom ide o povinný atribút, ktorý sa nesmie vynechať. Ak *use* neuvedieme vôbec, potom je to rovnaké, ako keby sme jeho hodnotu nastavili na „optional“ – je to východisková (t. j. implicitne nastavená) hodnota.

Jazyk XSD nám ďalej umožňuje pre každý atribút stanoviť *východiskovú hodnotu* (t. j. hodnotu, ktorá sa použije, ak v XML dokumente nebude uvedená iná hodnota pre tento atribút). Východiskovú hodnotu nastavíme prostredníctvom atribútu *default*, ktorý je potrebné uviesť do zodpovedajúceho elementu *attribute*. Nasledujúci riadok kódu ukazuje použitie elementu *attribute* a jeho atribútov *name*, *type*, *use* a *default* na definovanie atribútu s názvom *Mena*:

```
<xs:attribute name="Mena" type="xs:string" use="required" default="EUR"/>
```

Pri definovaní atribútov môžu nastať štyri typy situácií vyplývajúce z ich želanej pozície v XML dokumente [HER12]:

1. *Atribút má byť súčasťou elementu, ktorý má okrem neho obsahovať aj jeden alebo viacero vnorených elementov* – atribúty sa musia definovať až po definovaní všetkých vnorených elementov, t. j. až za koncovou značkou všetkých elementov *sequence*, *all* alebo *choice*. Toto pravidlo je častým zdrojom chýb, pretože v XML dokumentoch bývajú atribúty súčasťou začiatočných značiek elementov, a to u niektorých tvorcov XSD kódov vyvoláva mylný dojem, že by v XML schéme mali byť definované najskôr atribúty a až potom vnorené elementy. Správne riešenie tejto situácie demonštruje nasledovný úryvok kódu XML schémy:

```
<xs:complexType name="Tovar">
    <xs:sequence>
        <xs:element name="Názov_tovaru" type="xs:string"/>
        <xs:element name="Jednotková_cena" type="xs:float"/>
        <xs:element name="Opis" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="ID_tovaru" type="xs:ID" use="required"/>
</xs:complexType>
```

Vzhľadom na túto XML schému je validný napr. tento XML dokument:

```
<Tovar ID_tovaru="c114">
    <Názov_tovaru>Jogurt Rajo čučoriedkový</Názov_tovaru>
    <Jednotková_cena>0.45</Jednotková_cena>
    <Opis>Mliečny produkt</Opis>
</Tovar>
```

2. *Atribút má byť súčasťou elementu obsahujúceho hodnotu* – aj v tomto prípade takýto element v XML schéme označíme ako *complexType*. Do neho vnoríme element *simpleContent* vyjadrujúci skutočnosť, že ide o element, ktorý vo svojom vnútri neobsahuje vnorené elementy, ale medzi jeho začiatočnou a koncovou značkou sa smie nachádzať vždy iba jedna hodnota. Vo vnútri elementu *simpleContent* musí byť element

*extension* s atribútom *base*. Tento atribút vyjadruje dátový typ hodnoty nachádzajúcej sa medzi začiatočnou a koncovou značkou nami definovaného elementu (t. j. elementu, ktorý sme označili ako *complexType*). Do vnútra elementu *extension* treba uviesť element *attribute* s atribútm *name* a *type* a prípadne aj *use* a *default*, ktoré bližšie spezifikujú vlastnosti nami definovaného atribútu patriaceho zloženému elementu. Tento pomerne zložitý postup pri definovaní jednoduchého elementu obsahujúceho atribút demonštruje nasledujúci kód XML schémy:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <!-- Vytvorenie jednoduchého (enumeračného) používateľsky deklarovaného dátového
        typu s názvom MenaType-->
    <xs:simpleType name="MenaType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="EUR"/>
            <xs:enumeration value="USD"/>
            <xs:enumeration value="CZK"/>
        </xs:restriction>
    </xs:simpleType>

    <!-- Vytvorenie zloženého používateľsky deklarovaného dátového typu s názvom Jed-
        notkováCena obsahujúceho atribút typu MenaType-->
    <xs:element name="JednotkováCena">
        <xs:complexType>
            <xs:simpleContent>
                <xs:extension base="xs:decimal">
                    <xs:attribute name="Mena" type="MenaType" use="required"/>
                </xs:extension>
            </xs:simpleContent>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Príkladom XML dokumentu, ktorý je validný vzhľadom na túto XML schému, je tento krátky dokument:

```
<?xml version="1.0" encoding="UTF-8"?>
<JednotkováCena Mena="USD">85</JednotkováCena>
```

3. Atribút má byť súčasťou prázdnego elementu (t. j. elementu, pre ktorý platí, že medzi jeho začiatočnou a koncovou značkou sa nenachádza žiadna hodnota ani žiadny vnorený element) – takýto element opäť označíme ako *complexType* a do neho vnoríme element *attribute*. Elementy *simpleContent* a *extension* sa teda na rozdiel od predchádzajúceho prípadu vynechávajú. Nasledujúci kód je príkladom definície prázdnego elementu *Súbor* a jeho atribútu *Odkaz*:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsschema xmlns:xss="http://www.w3.org/2001/XMLSchema">
<xselement name="Súbor">
<xsccomplexType>
<xseattribute name="Odkaz" type="xss:anyURI" use="required"/>
</xsccomplexType>
</xselement>
</xsschema>
```

Uvedenej XML schéme zodpovedá napr. tento XML dokument:

```
<?xml version="1.0" encoding="UTF-8"?>
<Súbor Odkaz="D:/Pracovný_adresár/Zoznam_pracovníkov.docx">
</Súbor>
```

4. Viaceri atribútov v tom istom elemente – všetky atribúty patriace konkrétnemu elementu v XML dokumente sa v XML schéme uvádzajú postupne za sebou prostredníctvom elementov *attribute* vnorených do elementu *complexType*. Na ich poradí nezáleží, t. j. pri validácii XML dokumentu vzhľadom na konkrétnu XML schému sa nekontroluje poradie atribútov (na rozdiel od elementov, ktorých poradie sa kontroluje, ak nie sú súčasťou konštrukcie *all*). Napriek tomu je z dôvodu prehľadnosti a lepšej

orientácie vhodné zostavovať XML dokumenty a XML schémy tak, aby poradie atribútov v nich navzájom korešpondovalo. V nasledujúcej schéme definujeme element *Tovar* obsahujúci vnorené elementy *ID\_tovaru*, *Názov\_tovaru* a *Jednotková\_cena* a atribúty *Mena* a *DPH*:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="Tovar">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="ID_tovaru" type="xs:ID"/>
            <xs:element name="Názov_tovaru" type="xs:string"/>
            <xs:element name="Jednotková_cena" type="xs:decimal"/>
        </xs:sequence>
        <xs:attribute name="Mena" type="string" use="required"/>
        <xs:attribute name="DPH" type="decimal" use="required"/>
    </xs:complexType>
</xs:element>
</xs:schema>
```

XML dokument, ktorý je validný vzhľadom na túto XML schému, môže vyzeráť napr. takto:

```
<?xml version="1.0" encoding="UTF-8"?>
<Tovar Mena="EUR" DPH="0.19">
    <ID_tovaru>R011</ID_tovaru>
    <Názov_tovaru>Sveter pánsky tyrkysový</Názov_tovaru>
    <Jednotková_cena>9.99</Jednotková_cena>
</Tovar>
```

Ak si želáme do konkrétneho .xml súboru uviesť informáciu o tom, podľa ktorej XML schémy (t. j. podľa ktorého .xsd súboru) má byť tento dokument validovaný, môžeme to vykonať pomocou atribútov *noNamespaceSchemaLocation* alebo *SchemaLocation*, ktoré sa zapisujú do koreňového elementu XML súboru. To, ktorý z týchto atribútov pritom použijeme,

závisí od toho, či XML schéma, na ktorú sa odkazujeme, obsahuje cieľový menný priestor (t. j. *targetNamespace*). Ak ho obsahuje, potom sa používa atribút *SchemaLocation*, do ktorého uvedieme názov cieľového menného priestoru. Za ním nasleduje medzera a potom adresa umiestnenia .xsd súboru a jeho názov. Ak sa .xsd súbor nachádza v tom istom adresári ako .xml súbor, ktorý má byť podľa neho validovaný, potom stačí uviesť iba názov .xsd súboru (t. j. adresu umiestnenia súboru môžeme vynechať).

Ak XML schéma neobsahuje cieľový menný priestor, potom sa používa atribút *noNameSpaceSchemaLocation*. Do neho sa uvádzajú adresa umiestnenia .xsd súboru a jeho názov. Adresa umiestnenia sa pritom môže opäť vynechať, ak sa tento súbor nachádza v rovnakom adresári ako .xml súbor, ktorý sa má podľa neho validovať. Mnohé XML validátory sú však schopné validovať XML dokumenty vo vzťahu k XML schémam aj bez takého explicitného priradenia – stačí zvoliť oba súbory alebo nakopírovať ich kódy do príslušného okienka a validátor vykoná svoju prácu.

Príkladom použitia atribútu *noNameSpaceSchemaLocation* je nasledujúci úryvok XML kódu.

```
<Koreňový_element  
noNamespaceSchemaLocation="https://www.w3schools.com/xml/note.xsd.">  
...  
</Koreňový_element>
```

Príkladom použitia atribútu *SchemaLocation* je tento úryvok XML kódu (predpokladáme pritom, že v príslušnom .xsd súbore sa nachádza cieľový menný priestor s názvom *Škola*):

```
<Koreňový_element  
SchemaLocation="Škola D:\Škola\XML\XMLschémy\naša_schéma.xsd">  
...  
</Koreňový_element>
```

Vhodným a bezplatným on-line nástrojom na validáciu XML dokumentov podľa konkrétnych XML schém (.xsd súborov) je *W3C XML Schema (XSD) Validation on-line*, ktorý

je dostupný na webovej stránke: <http://www.utilities-online.info/xsdvalidation/#.%20WZCg19JJYdU>.

### 6.3 WSDL

Hlavným dokumentom opisujúcim určitú službu v SOA býva zvyčajne dokument zapísaný v jazyku **WSDL** (*Web Services Description Language*, t. j. „*Jazyk na opis webových služieb*“). Ak chce služba A zaslať určitú správu službe B, musí mať táto správa tvar zodpovedajúci tomu, čo je uvedené vo WSDL dokumente opisujúcim službu B. Naopak, ak chce služba B poslať správu s odpoveďou službe A, musí aj táto služba prispôsobiť štruktúru a obsah svojej správy tomu, čo je uvedené vo WSDL dokumente opisujúcim službu A.

**WSDL dokument** je základný dokument opisujúci webovú službu. Každý WSDL dokument musí byť štruktúrovaný ako XML dokument (t. j. musia v ňom byť dodržané všetky pravidlá pre tvorbu správne štruktúrovaných XML dokumentov). Ako sme uviedli v kapitole 3, webovú službu možno chápať ako rozhranie k určitej naprogramovanej funkciałite. Oxford Dictionary of Computing definuje pojem **rozhranie** ako „*spoločnú hranicu medzi dvoma zariadeniami, systémami alebo programami*“ [OXF08]. Claus a Schwill k problematike rozhrani uvádzajú, že rozhranie je použiteľné iba vtedy, ak sú známe všetky veličiny, ktoré je potrebné do rozhrania zadať zvonka (vtedy hovoríme o vstupnom rozhraní, resp. o **vstupnej časti rozhrania**) a všetky veličiny, ktoré má toto rozhranie poskytnúť okoliu (výstupné rozhranie, resp. **výstupná časť rozhrania**) [CLA91]. Inými slovami, pre každú službu musia byť známe jej vstupy a výstupy a tieto vstupy a výstupy musia byť definované vo WSDL dokumente, ktorý túto službu opisuje. Vďaka tomu sa o nich môžu dozviedieť jej používateľia, či už ide o ľudí alebo o počítačové programy. V tejto súvislosti však musíme zdôrazniť, že WSDL dokumenty sú určené primárne pre počítačové programy – sú písané v umelom jazyku preto, aby boli strojovo čitateľné.

WSDL dokument má **dve základné časti**:

1. **Abstraktný opis služby** – táto časť opisu špecifikuje **vstupné správy**, ktoré môžu byť tej-ktorej službe zasielané, a **výstupné správy**, ktoré môže táto služba zasielať iným službám. Správy sú zoskupované do tzv. *operácií* a operácie sú zasa združené do *rozhrania* (*portType*, resp. *interface*). Dôležité je, že abstraktný opis rozhrania neuvádzza technické detaily, na základe ktorých by bolo možné uskutočniť reálny prenos údajov a službu kontaktovať. Tie sú uvedené až v *konkrétnom opise služby*, ktorý vychádza

z informácií uvedených v abstraktnom opise. Oddelením týchto informácií sa dá ochrániť integrita opisu služby bez ohľadu na zmeny, ktorými môže prejsť technologická platforma, na ktorej je táto služba prevádzkovaná. Inými slovami, vďaka abstraktnému opisu *môžeme službu opísat' nezávisle od technológií, pomocou ktorých je implementovaná*. Ako sme už uviedli, v abstraktnom opise sa stretávame s troma základnými pojmom, ktorými sú:

- a. **Správa** (z angl. message) – správou rozumieme množinu súvisiacich údajov, ktoré do služby vstupujú (vstupná správa) alebo z nej vystupujú (výstupná správa). Tieto údaje sú zasielané ako jeden celok, pričom zasielanie správ je spôsob, akým spoľu služby komunikujú. V jazyku WSDL sa každá správa definuje prostredníctvom samostatného elementu *message*. V tomto elemente sa nenachádza informácia o tom, či ide o vstupnú alebo výstupnú správu, ale je tu špecifikovaný názov tejto správy. Do elementu *message* sa vnára element *part*, v ktorom sa pomocou atribútu *element* špecifikuje názov elementu, ktorý má táto správa obsahovať. Pomocou atribútu *type* sa tu môže špecifikovať tiež jeho dátový typ. Dátový typ však možno špecifikovať aj v samostatnom elemente *types*, ktorý sa vo WSDL dokumente nachádza mimo elementu *message* (t. j. ešte pred ním). Ak majú byť súčasťou správy viaceré elementy, potom sa do elementu *message* vnára viacero elementov *part*. Elementy v správach sú vlastne nositeľmi údajov – každý element reprezentuje určitý konkrétny údaj prenášaný v správe a názov elementu, v ktorom je tento údaj zapuzdrený, opisuje jeho význam, resp. čo tento údaj znamená/symbolizuje. Elementy s údajmi môžu obsahovať aj atribúty, ktoré o nich nesú ďalšie údaje.
- b. **Operácia** (z angl. operation) – mechanizmus výmeny správ jednej služby s inou sa v jazyku WSDL označuje ako **operácia**. Výmena správ pritom môže byť jednosmerná (odosielateľ správy nežiada odpoved') alebo obojsmerná (odosielateľ správy žiada odpoved').

Rozlišujeme štyri typy operácií:

- *prijatie vstupnej správy,*
- *prijatie vstupnej správy a zaslanie výstupnej správy s odpovedou,*
- *zaslanie výstupnej správy,*
- *zaslanie výstupnej správy a prijatie vstupnej správy s odpovedou.*

Operácia sa vo WSDL dokumentoch označuje pomocou elementu *operation*. V jeho vnútri sa môžu nachádzať elementy *input* a *output* (bud' iba jeden z nich alebo oba – v závislosti od toho, o ktorý z vyššie uvedených štyroch typov operácií ide). Práve elementy *input* a *output* určujú, či je niektorá správa z pohľadu služby vstupnou správou alebo výstupnou.

Každá operácia musí mať svoj vlastný názov, ktorý sa špecifikuje v atribúte *name* elementu *operation*. Elementy *input* a *output* majú zasa atribút *message*, v ktorom treba špecifikovať názov tej-ktorej vstupnej, resp. výstupnej správy. Dôležité pritom je, že v elemente *operation* sa môžeme odkazovať iba na správy, ktoré už boli vo WSDL dokumente skôr definované pomocou samostatného elementu *message* a tieto správy môžeme vo vzťahu ku konkrétnej operácii označiť ako vstupné alebo výstupné. Môžeme sa teda odkazovať len na správy, ktorých názvy a štruktúra (t. j. názvy a dátové typy elementov ako nositeľov údajov) sú už známe z predchádzajúcich častí WSDL dokumentu.

- c. **Rozhranie** (z angl. *portType*, resp. *interface*) – každá webová služba môže zabezpečovať vykonávanie jednej alebo viacerých operácií. Všetky tieto operácie sú súčasťou rozhrania služby, a preto musia byť vo WSDL dokumente, ktorý ju opisuje, ohraničené elementom *portType* reprezentujúcim toto rozhranic. Verzia 1.1 špecifikácie jazyka WSDL podporovala možnosť, že jedna služba mohla mať viaceré rozhrania. Od verzie 2.0 tejto špecifikácie však môže mať každá služba iba jedno rozhranie. Okrem toho táto špecifikácia zaviedla terminologickú zmenu – namiesto pojmu *portType* sa má používať pojmom *interface* [MAR09].
2. **Konkrétny opis služby** – abstraktný opis služby sám o sebe nepostačuje na to, aby sme s touto službou mohli reálne komunikovať. Technické detaily, ktoré sú na realizáciu tejto komunikácie potrebné, sú uvedené až v konkrétnom opise rozhrania.

Základnými pojмami, s ktorými sa stretávame v konkrétnom opise rozhrania, sú *väzba*, *port* a *služba*:

- a. **Väzba** (z angl. *binding*) – väzba vyjadruje vzťah rozhrania služby ku konkrétnym prenosovým *protokolom* a tiež k *protokolom vyjadrujúcim formát prenásaných správ*. Medzi protokoly vyjadrujúce formát správ patria najmä SOAP

(*Simple Object Access Protocol*) vo verzii 1.1 alebo 1.2 a MIME (*Multi-Purpose Internet Mail Extension*). Na prenos výlučne textových údajov sa zvykne používať protokol SOAP verzie 1.1., zatiaľ čo protokol MIME a verzia 1.2 protokolu SOAP sa používajú vtedy, ak majú byť súčasťou správy aj netextové údaje (napr. obrázky, zvukové nahrávky, obrazové nahrávky či rôzne iné typy súborov). Protokoly vyjadrujúce formát správ kladú základné požiadavky na to, ako majú byť tieto správy štruktúrované, resp. zapísané.

Prenosom správ sa zaoberajú *prenosové protokoly*, ktoré zasa špecifikujú pravidlá, ako majú byť tieto správy prenášané po sieti. Najčastejšie používaným prenosovým protokolom na prenos správ je HTTP (*Hypertext Transfer Protocol*), no dajú sa použiť aj protokoly SMTP (*Simple Mail Transfer Protocol*), FTP (*File Transfer Protocol*) a iné .

Vo väzbe (t. j. „binding“) sa ďalej špecifikuje tzv. *štýl správy*. Ak operácia, ku ktorej sa príslušná väzba vzťahuje, pozostáva zo vstupnej a aj z výstupnej správy, potom sa štýl správy zvykne špecifikovať zvlášť pre každú z týchto správ. Ak sa väzba vzťahuje naraz na viacero operácií, potom sa štýl správy stanovuje samostatne pre každú správu každej operácie. Rozlišujeme pritom štyri hlavné štýly správ, a to *RPC/encoded*, *RPC/literal*, *document/encoded* a *document/literal*, ktoré bližšie charakterizujeme v kapitole 6.5 *SOAP*.

Každá väzba sa vo WSDL dokumente zapisuje prostredníctvom samostatného elementu *binding*, pričom musí mať svoj unikátny názov (ten sa zapisuje do atribútu *name* v tomto elemente). Ku každému rozhraniu (*portType*) sa musí vytvoriť aspoň jedna väzba (*binding*). Okrem tejto podmienky počet väzieb nie je limitovaný. Musí však platiť, že v každej väzbe sú špecifikované všetky operácie rozhrania, t. j. počet operácií v jednej väzbe sa musí rovnať počtu operácií v jednom rozhraní. Okrem toho musí platiť, že názvy operácií uvedených vo väzbe sa musia zhodovať s názvami operácií uvedených v rozhraní.

- b. Port** – port je styčný bod, na ktorom sa dá kontaktovať server. Webová služba býva prevádzkovaná na serveri, ktorý spravidla poskytuje svojmu okoliu možnosť kontaktovať ho prostredníctvom viacerých takýchto styčných bodov. Porty bývajú číslované od 0 do 65635, pričom čísla od 0 do 1024 bývajú rezervované pre niektoré špecifické protokoly (napr. port 80 sa používa

pre komunikáciu prostredníctvom protokolu HTTP, port 25 pre komunikáciu cez protokol SMTP a pod. – čísla portov sa však na konkrétnom serveri dajú v prípade potreby zmeniť). Ak máme napr. server, ktorý je dostupný na URL adrese <http://xxx.yyy.com>, a chceme tento server kontaktovať na jeho portu číslom 918, potom to môžeme vykonať pomocou zápisu v tvare <http://xxx.yyy.com:918>. URL adresa servera, na ktorom je prevádzkovaná určitá služba, a číslo portu predstavuje *fyzickú adresu služby*. Ak číslo portu neuvedieme, potom sa použije na prenos údajov východiskový (t. j. „defaultný“) port (spravidla ide o port 80).

Vo verzii 2.0 špecifikácie WSDL sa označenie *port* zmenilo na *endpoint* (v preklade „koncový bod“). Význam tohto označenia sa však nezmenil. Každý port sa vo WSDL dokumente zapisuje pomocou samostatného elementu *port* (resp. *endpoint* vo verzii 2.0 špecifikácie WSDL) a musí mať svoj jedinečný názov, ktorý sa špecifikuje v atribútcii *name* tohto elementu. Ku každej väzbe (t. j. „binding“) sa špecifikuje jeden port, pričom ten istý port sa môže špecifikovať pre viaceré väzby. Počet elementov *port* sa vo WSDL dokumente každopäťne musí rovnať počtu elementov *binding*.

- c. **Služba** (z angl. *service*) – všetky porty, pomocou ktorých môžeme kontaktovať službu prevádzkovanú na určitom serveri, sa vo WSDL dokumente zoskupujú do elementu *service* (t. j. sú do neho vnorené ako jeho detské elementy).

Okrem elementov spomínaných vyššie sa vo WSDL dokumentoch zvyknú vyskytovať aj iné elementy, a to:

- **definitions** – predstavuje koreňový element WSDL dokumentu. To znamená, že všetky ostatné elementy v tomto dokumente musia byť vnorené do tohto elementu – s výnimkou hlavičkového elementu s údajmi o verzii XML a kódovaní znakov v dokumente (t. j. napr.: `<?xml version="1.0" encoding="UTF-8" ?>`). Keďže každý WSDL dokument je zároveň XML dokumentom, údaje o kódovaní a verzii XML by sa nemali vynechať.
- **types** – do tohto elementu môžu byť vnorené XML schémy, v ktorých sú špecifikované dátové typy elementov a atribútov, ktoré majú byť prenášané vo vstupných a výstupných správach. V elemente *types* sa pritom môže vyskytovať viacero schém

s tým, že každá schéma musí mať svoj vlastný rodičovský element *schema* a musí byť zapisaná v súlade s pravidlami jazyka XSD.

- *documentation* – tento element sa používa na zápis komentárov a môže sa objavíť vo vnútri akéhokoľvek iného elementu vo WSDL dokumente. Zmyslom používania komentárov je zabezpečiť to, aby boli WSDL dokumenty lepšie čitateľné a zrozumiteľné pre človeka (kedže jazyk WSDL je pomerne zložitý a dokumenty písané v ňom sú primárne určené pre strojové spracovanie, t. j. spracovanie na počítači).
- *input* – tento element sa používa na pripojenie externých XML schém (t. j. .xsd súborov) k WSDL dokumentu. To znamená, že ak schéma nie je priamo súčasťou WSDL dokumentu, môžeme ju k nemu pripojiť pomocou elementu *input*, čím dosiahneme rovnaký výsledok, ako keby bol kód tejto schémy zapisaný v elemente *types*, spomínanom vyššie.

WSDL dokumenty by mali mať nasledovnú štruktúru [W3C01]:

```
<definitions>
    <!--Element types nie je povinný. -->
    <types></types>
    <!--Každá vstupná alebo výstupná správa sa definuje pomocou
        samostatného elementu <message>. -->
    <message></message>

    <!--V elemente portType sa môže nachádzať ľubovoľný počet operácií.
        Pre každú operáciu treba samostatný element operation. -->
    <portType>
        <operation></operation>
        <operation></operation>
    </portType>

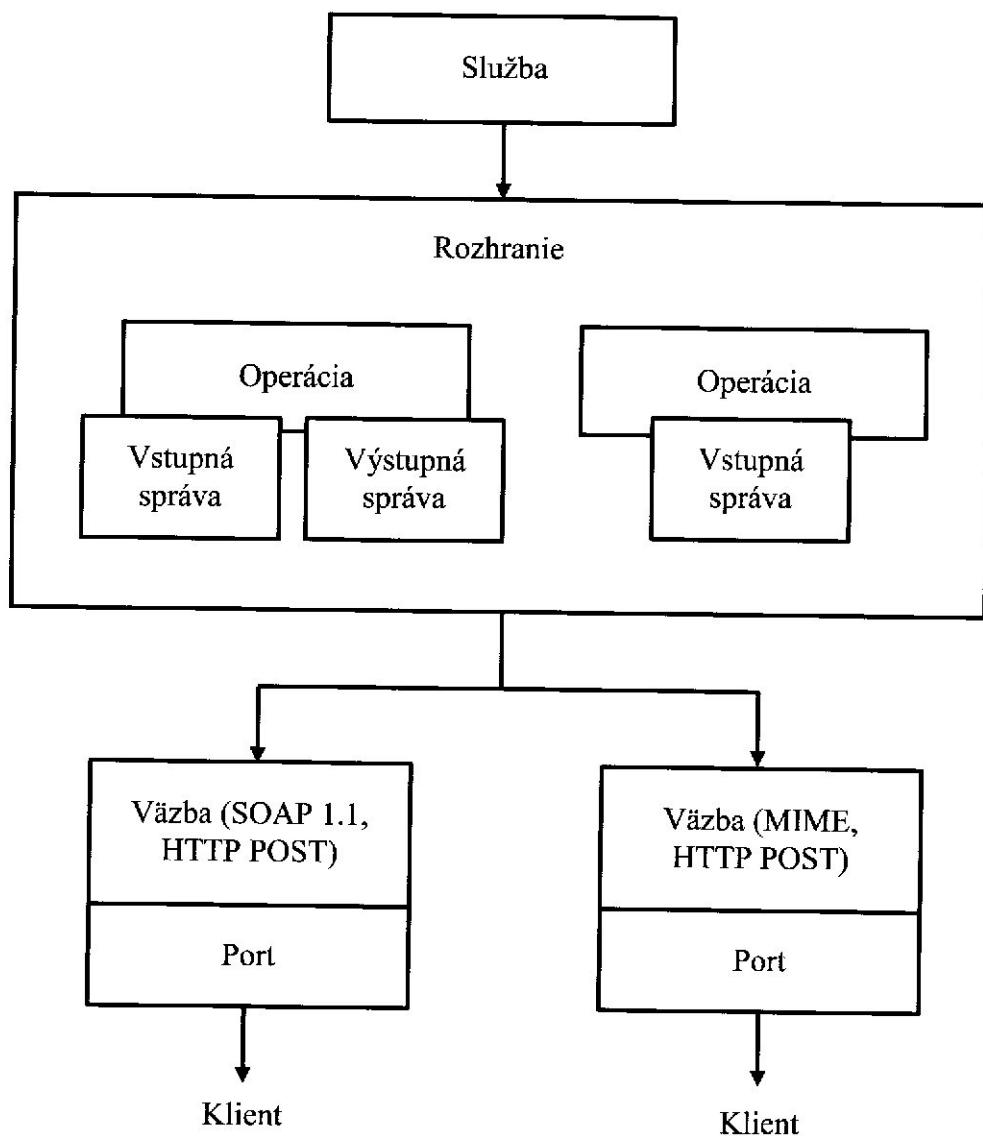
    <!--Pre každé rozhranie (portType) sa musí špecifikovať minimálne jedna väzba
        (binding). -->
    <!--Počet operácií v jednej väzbe sa musí rovnať počtu operácií v rozhraní a ich názvy
        sa musia zhodovať. -->
```

```

<binding>
    <operation></operation>
    <operation></operation>
</binding>

<!--Porty sú združené do elementu service.-->
<!--Počet elementov port v službe sa pritom musí rovnať počtu elementov binding.-->
<service>
    <port></port>
</service>
</definitions>

```



Obr. 35: Vzťahy medzi jednotlivými prvками WSDL špecifikácie [Zdroj: autor]

Príkladom WSDL dokumentu je nasledujúci kód. V tomto dokumente opisujeme službu „Banka“, ktorá svojmu okoliu sprístupňuje jednu operáciu. Táto operácia pozostáva z prijatia vstupnej správy a následného poskytnutia výstupnej správy. Vstupnou správou je žiadosť o poskytnutie údajov o konkrétej banke a výstupnou správou sú požadované údaje adresované ich žiadateľovi. Predpokladáme teda, že ide o entitnú službu „Banka“, ktorá je schopná pristupovať do databázy o bankách a na vyžiadanie z tejto databázy poskytovať rozličné údaje. V tomto WSDL dokumente sa pritom zaoberáme len funkcionálou typu READ (t. j. pre jednoduchosť abstrahujeme od funkcionálít typu CREATE, UPDATE a DELETE, ktorých začlenenie do riešenia by navýšilo celkový počet operácií zabezpečovaných službou na štyri).

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
    targetNamespace=" http://www.abc.com/services/Banka">
```

<documentation> Atribút xmlns="http://schemas.xmlsoap.org/wsdl/" špecifikuje východiskový menný priestor, t. j. menný priestor vzťahujúci sa na elementy, ktoré nie sú označené prefixom žiadneho iného menného priestoru. Východiskový menný priestor nemá prefix. Špecifický je tiež menný priestor označený ako *targetNamespace*. Ten združuje všetky menné priestory nachádzajúce sa v jednom WSDL dokumente – vrátane východiskového menného priestoru. </documentation>

<documentation> V elemente *types* sa nachádza jedna XML schéma, ktorá definuje dátový typ pre element *NázovBanky* (ide o typ *string* ako jeden zo základných dátových typov) a tiež dátový typ pre element *Odpoved* (zložený používateľsky deklarovaný dátový typ).</documentation>

```
<wsdl:types>
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```

<xsd:element name="NázovBanky" type="xsd:string"/>
<xsd:element name="Odpoved">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="IČO" type="xsd:string"/>
            <xsd:element name="IČDPH" type="xsd:string"/>
            <xsd:element name="Kód banky" type="xsd:string"/>
            <xsd:element name="Označenie banky" type="xsd:string"/>
            <xsd:element name="SWIFT kód" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
</xsd:schema>
</wsdl:types>

```

<documentation>V elemente *message* definujeme správu s názvom *ZadajBanku*, ktorá má obsahovať element s názvom *NázovBanky*. Pomocou tohto názvu žiadateľ službe špecifikuje, o ktorej banke potrebuje získať údaje.

</documentation>

```

<message name="ZadajBanku">
    <part element="NázovBanky"/>
</message>

```

<documentation>V elemente *message* definujeme správu s názvom *PoskytniÚdajeOBanke*, ktorá má obsahovať element s názvom *Odpoved*. Do takto nazванého elementu budú vo výstupných správach zapisované všetky požadované údaje podľa štruktúry definovanej v elemente *types*.

```

<message name="PoskytniÚdajeOBanke">
    <part element="Odpoved"/>
</message>

```

<documentation>V elemente *portType* definujeme rozhranie s názvom *Rozhranie* pozostávajúce z jednej operácie, ktorou je operácia *ÚdajeOBanke*. Táto operácia pozostáva zo vstupnej správy s názvom *ZadajBanku* a výstupnej správy s názvom *PoskytniÚdajeOBanke*.

```

</documentation>

<portType name="Rozhranie">
    <operation name="ÚdajeOBanke">
        <input message="ZadajBanku"/>
        <output message="PoskytniÚdajeOBanke"/>
    </operation>
</portType>

```

<documentation>Na tomto mieste sa končí abstraktný opis rozhrania a začína sa jeho konkrétny opis. </documentation>

<documentation>Pre *Rozhranie* definujeme dve väzby s názvami *Väzba1* a *Väzba2*. Prvá väzba určuje, že vstupná a výstupná správa, ktoré tvoria operáciu *ÚdajeOBanke*, majú byť naformátované podľa protokolu SOAP verzie 1.1. Druhá väzba týmto správam zasa predpisuje protokol SOAP verzie 1.2. Vznikajú tak dve akceptovateľné alternatívy, ako môžu byť správy naformátované. V obidvoch väzbách sa požaduje, aby boli vstupné aj výstupné správy štruktúrované v štýle document/literal (o tomto štýle a iných alternatívnych štýloch si viac povieme v kapitole 6.5 *SOAP*) a aby boli zasielané pomocou prenosového protokolu HTTP. Ak chceme použiť práve tento protokol, musíme atribút *transport* inicializovať URL adresou: <http://schemas.xmlsoap.org/soap/http>. Atribút *soapAction* je povinný iba pri použití protokolu SOAP 1.2 (Pri protokole SOAP 1.1 je nepovinný – ak ho vynecháme, použije sa jeho východisková hodnota – prázdný reťazec znakov). Ak je do tohto atribútu uložený prázdný reťazec, znamená to, že služba, ktorá prijme zodpovedajúcu správu, nájde všetky informácie, ktoré potrebuje na to, aby vykonala správnu operáciu, priamo v tejto správe. Ak by sa nejaké dodatočné informácie nachádzali na externej URL adrese, zadali by sme do atribútu *soapAction* práve túto adresu. </documentation>

```

<binding name="Väzba1" type="Rozhranie">
    <soap:binding style="document"
                  transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="ÚdajeOBanke">
        <soap:operation style="document"/>
        <input>
            <soap:body use="literal"/>

```

```

</input>
<output>
    <soap:body use="literal"/>
</output>
</operation>
</binding>

<binding name="Väzba2" type="Rozhranie">
    <soap12:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="ÚdajeOBankc">
        <soap12:operation soapAction="" style="document"/>
        <input>
            <soap12:body use="literal"/>
        </input>
        <output>
            <soap12:body use="literal"/>
        </output>
    </operation>
</binding>

<documentation>V elemente service špecifikujeme názov služby opísanej vo WSDL dokumente (ako sme už spomenuli, ide o entitnú službu s názvom „Banka“). Rozhranie tejto služby môžu klienti kontaktovať prostredníctvom dvoch portov, ktorými sú: Port1 vzťahujúci sa k Väzbe1 a Port2 vzťahujúci sa k Väzbe2. Každý z týchto portov musí mať špecifikovanú svoju fyzickú adresu, a to sa vykonáva pomocou elementu soap:address (v prípade protokolu SOAP 1.1) alebo pomocou elementu soap12:address (v prípade protokolu SOAP 1.2). Keďže v obidvoch prípadoch sa používa prenosový protokol HTTP (to je špecifikované v definíciách Väzby1 a Väzby2), môže ísť o ten istý port.
</documentation>

<service name="Banka">
    <port name="Port1"
        binding="Väzba1">

```

```

<soap:address location="http://www.abc.com/services/Banka:80"/>
</port>

<port name="Port2"
      binding="Väzba2">
    <soap12:address location="http://www.abc.com/services/Banka:80"/>
  </port>
</service>

<documentation>Koncová značka elementu definitions zakončuje WSDL dokument.
</documentation>
</definitions>

```

Ako sme už uviedli, úlohou WSDL dokumentu je opísať, akým spôsobom môžeme kontaktovať určitú službu. Špecifíkujú sa tu operácie podporované webovou službou, formáty vstupných a výstupných správ, protokoly na ich prenos a adresy portov, prostredníctvom ktorých môžu klienti službu kontaktovať. Nič sa tu však nehovorí o tom, aké výpočty, resp. úkony musí táto služba vykonať na to, aby bola schopná transformovať vstupy na výstupy, resp. vykonať to, čo sa od nej požaduje. WSDL dokumenty teda skrývajú implementačné detaily služby (t. j. skrývajú, aké konkrétné funkcie sú na tejto službe vlastne spúšťané a aké sú ich algoritmy, resp. zdrojové kódy). Skrývanie implementačných detailov pred okolitým svetom je v informatike často používaný princíp, ktorý sa označuje ako **princíp čiernej skrinky**. Používa sa napr. aj v objektovo orientovanom programovaní, kde jednotlivé objekty, tvoriace objektovo orientovaný program, poskytujú svojmu okoliu informácie o tom, aké metódy (t. j. funkcie) poskytujú, aké vstupy tieto metódy potrebujú a aké poskytujú výstupy. Implementačné detaily (t. j. telá, resp. zdrojové kódy jednotlivých metód) však zostávajú pre okolie objektu skryté.

## **6.4 UDDI – registre služieb**

Ako sme už spomenuli v kapitole 6.3 WSDL, nutnou podmienkou, ktorá musí byť splnená na to, aby služba A mohla kontaktovať službu B je, že služba A musí mať prístup k dokumentu opisujúcemu službu B. Pri malom počte služieb môžu byť tieto služby naprogramované tak, aby vedeli o svojej vzájomnej existencii a aby mali všetky informácie potrebné na to, aby sa mohli vzájomne kontaktovať. S narastajúcim počtom služieb je však

*potrebné vytvoriť mechanizmus reprezentujúci centrálny adresár služieb, ktorého úlohou je poskytovať službám alebo používateľom možnosť vyhľadávať služby podľa nimi zadaných kritérií a sprístupňovať im aktuálne verzie opisov služieb.*

Takýto mechanizmus býva obvykle reprezentovaný registrom **UDDI** (*Universal Description, Discovery and Integration*). UDDI je platformovo nezávislý protokol pre registráciu webových služieb uchovávajúci údaje prostredníctvom XML súborov. Všetky služby zaregistrované v tomto registri sú vyhľadateľné a prístupné pre všetky iné služby, ktoré majú do neho prístup. Ku každej službe zaregistroanej v takomto registri musí byť priložený WSDL dokument, ktorý ju opisuje, a do ktorého sú v prípade potreby nahliadnuť všetky ostatné služby, aby vedeli, ako sa s vybranou službou skontaktovať. Okrem rozhrania pre služby mávajú UDDI registre spravidla rozhranie aj pre človeka – aby sa do takéhoto registra mohol pozrieť aj človek a v prípade potreby v ňom vyhľadať určitú službu.

Registre UDDI môžeme rozčleniť na **verejné** (t. j. „public“) a **súkromné** („private“). **Verejné registre** sa vyznačujú tým, že prístup do nich nie je obmedzený príslušnosťou ku konkrétnnej firme alebo inštitúcii (t. j. prístup môže mať každý, kto o to prejaví záujem). Hlavným zmyslom ich vytvárania bola snaha o podporu elektronického obchodu medzi firmami (t. j. služba jednej firmy mohla pomocou tohto registra automaticky vyhľadávať služby iných firm želaného podnikateľského zamerania a kontaktovať ich).

Štandard UDDI vznikol v roku 2000 so zámerom vytvoriť systém na podporu obchodnej spolupráce a výmeny údajov medzi rozličnými firmami, ktorý bude založený na používaní štandardizovaných webových služieb odstraňujúcich problémy so vzájomnou nekompatibilitou ich informačných systémov. Významné IT firmy, ako napr. IBM, Microsoft alebo SAP, investovali značné finančné prostriedky do vytvorenia vlastných verejných registrov služieb označovaných ako **UBR** (*UDDI Business Registry*). V roku 2006 však tieto firmy ohlásili zrušenie svojich verejných registrov, a to najmä z dôvodu nedostatočného záujmu poskytovateľov a žiadateľov služieb o takúto technológiu. Myšlienka verejných registrov sa teda veľmi neujala. Práca s verejnými registromi pritom prebichala takto:

1. Poskytovateľ webových služieb (firma alebo inštitúcia) sa zaregistrouje v registri UDDI (t. j. zaeviduje v registri všetky potrebné kontaktné údaje o svojej firme a tiež klasifikačné údaje, ktoré charakterizujú predmet jej podnikateľskej činnosti).
2. Poskytovateľ webových služieb následne do tohto registra zaregistrouje všetky služby, ktoré chce takouto formou poskytovať.

3. Žiadateľ služby (firma alebo inštitúcia) si na základe údajov o jednotlivých službách, zaregistrovaných v registri, a ich poskytovateľoch vyhľadá konkrétnu službu, ktorú chce kontaktovať. Register pritom môže prechádzat' buď človek (manuálne) alebo aplikácia/služba (automaticky).
4. Žiadateľ služby získá prístup k WSDL dokumentu opisujúcemu vybranú službu a v ňom nájde všetky údaje, ktoré sú potrebné na to, aby s ňou mohol skontaktovať.

Verejné registre boli typické svojou štruktúrou pozostávajúcou z troch hlavných častí, ktorými boli *biele stránky*, *žlté stránky* a *zelené stránky* [TUT20]:

- **Biele stránky** (*White Pages*) – v tejto časti verejného registra UDDI sa uchovávajú kontaktné údaje na jednotlivých poskytovateľov webových služieb, ktorí sú zaregistrovaní v tomto registri. Pre každého zaregistrovaného poskytovateľa služby tu musí byť uvedený údaj o názve príslušnej firmy alebo inštitúcie, ďalej údaje o jej adrese, ďalšie kontaktné údaje (email, telefón a pod.) a tiež unikátne identifikačné údaje (najmä IČO a DIČ a ich zahraničné ekvivalenty).
- **Žlté stránky** (*Yellow Pages*) – obsahujú klasifikačné údaje, ktoré charakterizujú podnikateľské zameranie firmami uvedených na bielych stránkach registra. Na základe žltých stránok musí byť teda o každej firme jasné, čomu sa príslušná firma venuje. Pri klasifikácii podnikateľského zamerania firmami sa využívajú štandardné klasifikačné taxonómie, medzi ktoré patria najmä *Standard Industrial Classification* (SIC), *North American Industry Classification System* (NAICS), *United Nations Standard Product and Services Code* (UNSPSC) a geografické taxonómie. *Taxonómie predstavujú zoznamy hierarchicky usporiadaných kategórií podnikateľských aktivít, pričom každá aktivita má priradené unikátne číslo v príslušnej taxonómii.* Ide teda o akési registre potenciálnych oblastí, v ktorých sa dá podnikať, pričom každá firma musí byť zaradená aspoň do jednej kategórie v každom registri (ak má široké portfólio aktivít a venuje sa viacerým oblastiam, môže byť zaradená naraz do viacerých kategórií).

Napr. *Výroba potravín* má v taxonómii SIC kód 20, pričom je rozdelená na množstvo hierarchicky usporiadaných podkategórií. Napr. *Mliečne výrobky* majú kód 202 a vnútorene sa členia na *Maslá* (kód 2021), *Prirodné, spracované syry a ich napodobeniny* (kód 2022), *Suché, kondenzované a odparované mliečne produkty* (kód 2023), *Zmrzliny a mrazené dezerty* (kód 2024) a *Tekuté mlieko* (kód 2026) [SIC87].

- **Zelené stránky (Green Pages)** – obsahujú údaje o tom, aké služby sprístupňujú poskytovatelia služieb prostredníctvom príslušného registra. Ku každej z týchto služieb je tu k dispozícii WSDL dokument, ktorý ju opisuje a hovorí, ako sa dá s touto ktorou službou po technickej stránke spojiť. Okrem údajov zapísaných vo WSDL dokumente tu môžu byť uvedené aj emaily, telefónne čísla či iné kontaktné údaje na osoby, ktoré môžu v prípade potreby poskytnúť o konkrétnej službe aj ďalšie doplňujúce informácie (t. j. dajú sa s nimi skonzultovať možnosti používania konkrétnej služby).

**Súkromné registre** bývajú vlastnené, spravované a kontrolované iba jednou firmou alebo inštitúciou a sú na rozdiel od verejných registrov určené na používanie iba vo vnútri tejto firmy (v rámci jej interného informačného systému). Pritom sa predpokladá, že táto firma alebo inštitúcia má v prevádzke informačný systém vybudovaný na princípoch SOA, t. j. systém pozostávajúci z množiny menších webových služieb, ktoré sa v prípade potreby vzájomne vyhľadávajú a kontaktujú na základe údajov dostupných v súkromnom registri. Za účelom rozšírenia možností obchodnej spolupráce môže byť prístup do súkromného registra rozšírený tak, aby do neho mohli pristupovať aj obchodní partneri tejto firmy. Súkromné registre UDDI sú teda na rozdiel od verejných registrov naďalej používajú a nestrácajú svoj význam.

Register UDDI uchováva údaje v XML súboroch, ktorých štruktúra je presne daná konkrétnymi XML schémami. Postupom času boli vytvorené 3 základné verzie registra UDDI, ktorých XSD sú zverejnené na webovej stránke: <https://msdn.microsoft.com/enus/library/ms953967.aspx>. Verzie 2.0 a 3.0 boli akceptované organizáciou *OASIS (Organization for the Advancement of Structured Information Standards)* ako štandard. Tieto verzie sa navzájom odlišujú povinnou štruktúrou XML súborov a niektorými technickými detailmi, no ich podstata a úloha zostávajú rovnaké. XML súbor v UDDI registri prislúchajúci ku konkrétnej službe musí obsahovať odkaz na zodpovedajúci WSDL súbor, no niektoré údaje z WSDL sú uchovávané aj priamo v elementoch a atribútoch tohto XML súboru.

## 6.5 SOAP

Vzhľadom na to, že všetka komunikácia medzi webovými službami v SOA je založená na posielaní a prijímaní správ, je potrebné tieto správy, ako aj ich prenos, štandardizovať, t. j. vytvoriť jasné a všeobecne akceptované pravidlá, ktoré je pri tom potrebné dodržiavať. Skratka SOAP pochádza z anglického slovného spojenia *Simple Object Access Protocol*, t. j. *Protokol*

*na jednoduchý rýchly prístup k objektom. Ide o formát, ktorý dodržiava všetky princípy jazyka XML a zapisujú sa v ňom správy prenášané medzi službami.*

Každá správa štruktúrovaná podľa formátu SOAP (ďalej len *SOAP správa*) musí byť „zabalená“ do obalu, ktorý sa nazýva *obálka*. **Obálka** (z angl. *envelope*) v sebe zapuzdruje celý obsah správy, ktorý je rozdelený do dvoch hlavných častí. Týmito časťami sú:

- **Hlavička** (z angl. *header*) – je časť správy, ktorá je vymedzená pre zápis *metadát*. Metadáta predstavujú doplnkové údaje, ktoré sa viažu k údajom prenášaným v hlavnej časti správy. Niekoľko sa preto označujú aj ako údaje o údajoch. Hlavička nie je povinnou súčasťou SOAP správy, ale zvyčajne sa nevynecháva, pretože sa do nej môžu zapisovať údaje dôležité z hľadiska doručenia správy a spracovávania jej obsahu cieľovými aplikáciami.
- **Telo** (z angl. *body*) – je hlavná časť správy. Nachádzajú sa v nej údaje, ktorých prenos je zmyslom posielania správy.

Jedným z cieľov pri budovaní systému na báze SOA je zabezpečiť čo najvyššiu nezávislosť prenášaných správ. Prostriedkom, ako to dosiahnuť, je vybaviť všetky správy dostatočným množstvom metadát potrebných na to, aby mohli byť nasmerované do cieľového bodu, kde ich spracuje príslušná aplikácia.

Každá SOAP správa musí byť zapísaná ako správne štruktúrovaný (well-formed) XML dokument (t. j. musia byť dodržané všetky pravidlá, ktoré sa kontrolujú pri validácii XML dokumentov). Obálka sa v SOAP správach definuje pomocou elementu *envelope*, ktorý je *koreňovým elementom správy*. Vo vnútri tohto elementu sa definuje hlavička pomocou elementu *header* (nepovinná časť správy) a telo pomocou elementu *body* (povinná časť). Element *header* pritom musí vždy predchádzať elementu *body*. V elementoch *header* a *body* sa nachádzajú dcérske, resp. vnorené elementy nesúce konkrétné údaje. Základná štruktúra SOAP správ teda vyzerá takto (atribút *xmlns* pritom reprezentuje východiskový menný priestor):

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">  
  <Header>  
    ...  
  </Header>  
  <Body>  
    ...
```

```
</Body>  
</Envelope>
```

V hlavičkách SOAP správ sa môžu vyskytovať rozličné elementy podľa špecifických potrieb aplikácie, ktorej sú tieto správy určené. Medzi často používané **elementy** patria:

- *Transaction* – udáva fyzickú adresu zdroja (URI), ku ktorému pristupuje určitá transakcia. Takýmto zdrojom môže byť napr. databáza alebo určitý súbor umiestnený na serveri. Príkladom použitia elementu *transaction* v hlavičke SOAP správy je nasledujúci kód. Hodnota 5 môže reprezentovať napr. ID transakcie (v závislosti od potrieb aplikácie, ktorej je táto správa určená). Menný priestor s prefixom *t* zasa určuje fyzickú adresu databázy, na ktorej sa má transakcia uskutočniť.

```
<Header>  
  <t:Transaction  
    xmlns:t="db:firebird://localhost//path/to/some.db"  
    mustUnderstand="0">  
    5  
  </t:Transaction>  
</Header>
```

- *CorrelationID* – udáva číslo správy, na ktorú je aktuálna správa odpoved'ou (slúži na sledovanie vzájomnej nadväznosti správ, t. j. jedna správa je odpoved'ou na inú správu). Napr. podľa nasledujúceho úryvku kódu je príslušná správa odpoved'ou na správu s číslom 014285464546-HGIJK65PL:

```
<Header>  
  <x:CorrelationID  
    xmlns:x="http://www.xmltc.com/tls/headersample"  
    mustUnderstand="1">  
    014285464546-HGIJK65PL  
  </x:CorrelationID>  
</Header>
```

V jednotlivých elementoch hlavičky sa môžu vyskytnúť tieto **atribúty**:

- *Actor* – SOAP správa putuje od svojho počiatočného odosielateľa ku konečnému príjemcovi, pričom na svojej ceste môže (ale nemusí) prechádzať cez jeden alebo viacero sprostredkovacích uzlov (služieb, ktoré rozhodujú o jej ďalšom smerovaní). Ako *sprostredkovatelia* sa v tomto kontexte označujú aplikácie (služby), ktoré sú schopné prijať SOAP správu a preposlať ju ďalej. Každý sprostredkovateľ ako aj konečný príjemca má svoju vlastnú fyzickú adresu (URI). Pokiaľ sú niektoré časti (t. j. elementy) hlavičky určené iba pre určitého sprostredkovateľa, potom musí byť príslušný element označený atribútom *actor*, do ktorého sa špecifikuje URI tohto sprostredkovateľa. Tento sprostredkovateľ potom správu preposiela ďalej bez tohto elementu. Ak v atribúte *actor* špecifikujeme URI "<http://schemas.xmlsoap.org/soap/actor/next>", dávame tým na znamosť, že zodpovedajúci element je určený pre prvú službu, ku ktorej sa táto správa dostane. Ak atribút *actor* vynecháme, potom je element primárne určený pre konečného príjemcu správy, no môžu si ho prečítať aj sprostredkovatelia [W3C01].
- *mustUnderstand* – ak tento atribút umiestníme do určitého elementu v hlavičke, môžeme pomocou neho stanoviť, či je príjemca správy (teda služba, ku ktorej sa táto správa dostane) povinný pri spracovávaní správy zohľadniť obsah tohto elementu, alebo je jeho zohľadenie iba voliteľné (t. j. nepovinné). Hodnota *mustUnderstand*=*"1"* značí, že ide o element, ktorý je pre príjemcu povinný, zatiaľ čo hodnota *mustUnderstand*=*"0"* označuje nepovinný element. Vynechanie tohto atribútu je ekvivalentné s použitím hodnoty *mustUnderstand*=*"0"*. Ak atribút *mustUnderstand* umiestníme priamo do elementu *header*, potom má tento atribút globálnu platnosť. To znamená, že sa vzťahuje na všetky elementy vnorené do elementu *header*. To isté platí aj o atribúte *actor*.

Služby, ktoré prijímajú a odosielajú SOAP správy sa niekedy označujú aj ako *uzly SOAP*. Existujú pritom viaceré typy takýchto uzlov [ERL09]:

- *odosielateľ SOAP* – uzol SOAP, ktorý vysiela správu,
- *prijemca SOAP* – uzol SOAP, ktorý prijíma správu,
- *sprostredkovateľ SOAP* – uzol SOAP, ktorý prijíma správu a preposiela ju inému uzlu s tým, že môže (ale nemusí) túto správu pred jej odoslaním spracovať (podľa toho ide o *aktívneho* alebo *pasívneho sprostredkovateľa*),
- *počiatočný odosielateľ SOAP* – prvý uzol SOAP, ktorý vysiela správu,
- *konečný príjemca SOAP* – posledný uzol, ktorý prijíma správu.

*Sprostredkovatelia SOAP* teda môžu vystupovať v pozícii príjemcov, ale aj odošielateľov SOAP správ. Cesta, ktorou správa prejde od chvíle, keď bola po prvýkrát odoslaná, do chvíle, keď sa dostane do konečného cieľa, sa nazýva *cesta správy*. Táto cesta sa skladá z jedného počiatočného odošielateľa, jedného konečného príjemcu a nijakého alebo niekoľkých sprostredkovateľov. Analyzovanie ciest správ má v SOA význam najmä preto, aby nedochádzalo k neefektívному smerovaniu správ cez množstvo sprostredkovateľov, čím by sa zbytočne predlžovalo dokončovanie procesov, ktoré od týchto správ závisia. Cesta správy pritom nemusí byť vždy daná vopred (t. j. nemusí byť *statická*). Môže byť určovaná aj *dynamicky* (t. j. priamo počas toho, ako správa putuje) na základe rozhodnutia jedného alebo viacerých sprostredkovateľov. Tí sa pritom rozhodujú na základe pravidiel, ktoré majú v sebe implementované, a údajov, ktoré sa nachádzajú v hlavičke správy.

Teraz sa zamerajme na telá SOAP správ. Ich štruktúra musí v prvom rade zodpovedať štýlu správy, ktorý je uvedený vo WSDL dokumente opisujúcom príslušnú službu. Názvy a dátové typy elementov, nachádzajúce sa v tele správy, ako aj poradie elementov musia takisto korešpondovať s údajmi uvedenými vo WSDL dokumente, príp. s údajmi uvedenými v externých XML schémach, ktoré sú k nemu pripojené. Pripomíname, že štýl SOAP správy sa vo WSDL dokumente definuje v konkrétnnej časti opisu služby, a to v sekcií *binding* pomocou atribútov *style* (môže nadobúdať hodnoty *RPC* alebo *document*) a *use* (môže nadobúdať hodnoty *encoded* alebo *literal* – vid' kapitola 6.3 *WSDL*). Existujú pritom štyri rozličné kombinácie hodnôt týchto dvoch atribútov (t. j. **štýri štýly SOAP správ**) [BUT05]:

- RPC/encoded,
- RPC/literal,
- Document/encoded,
- Document/literal.

Skratka **RPC** znamená **vzdialené volanie procedúr** (z angl. *Remote Procedure Call*). Štýl *RPC* sa používa vtedy, *ak poznáme presný názov funkcie, ktorú chceme spustiť na určitom serveri, a vieme tiež, aké vstupné údaje od nás táto funkcia potrebuje, aby mohla vykonáť činnosť, kvôli ktorej ju spúšťame*. Pri použití tohto štýlu musí byť v tele správy, ktorú zasielame cielovej službe, špecifikovaný názov funkcie, ktorú sa usilujeme spustiť. To znamená, že do elementu *body* musí byť vnorený *element s názvom tejto funkcie* (t. j. začiatočná aj koncová značka tohto elementu musia mať zhodný názov s funkciou, ktorú chceme spustiť).

V tejto súvislosti rozlišujeme *parametrické* a *bezparametrické funkcie*. **Bezparametrické funkcie** sú také, ktoré od subjektu, ktorý ich volá (môže to byť iná funkcia alebo používateľ), nepotrebuju žiadne vstupné údaje a stačí im len prijatie signálu (t. j. správy), že majú niečo vykonať. Pri volaní bezparametrickej funkcie preto do *elementu s názvom tejto funkcie nemusíme umiestňovať žiadne vstupné údaje*. To znamená, že do tohto elementu nemusia byť vnorené žiadne ďalšie elementy. Predpokladá sa totiž, že funkcia buď už disponuje všetkými potrebnými vstupmi, alebo si vie prístup k nim zabezpečiť sama (t. j. nepožaduje dodanie vstupov od subjektu, ktorý ju volá). *Pojem volanie funkcie pritom vyjadruje snahu o jej spustenie, ide teda o impulz na aktiváciu funkcie*.

**Parametrické funkcie**, na rozdiel od bezparametrických, požadujú, aby im subjekt, ktorý ich spúšťa, dodal určité vstupné hodnoty, ktoré potrebujú pre svoju činnosť. Pritom nemusí ísť o všetky vstupné hodnoty, ak si funkcia je schopná niektoré z nich zabezpečiť sama. V každom prípade musí telo správy obsahovať určité vstupné hodnoty, ktoré od nás funkcia očakáva. Tieto vstupné hodnoty bývajú reprezentované samostatnými *elementmi*, ktoré sa vnárajú do *elementu s názvom funkcie*. Pre každú vstupnú hodnotu sa zvykne vytvoriť samostatný element, do ktorého sa táto hodnota zapíše. Obsah správy sa ďalej odvíja od toho, či ide o štýl *RPC/encoded* alebo *RPC/literal* [ANG11].

Správa, určená na spustenie vzdialenej funkcie, vytvorená v štýle RPC môže mať napr. tento tvar [ANG11]:

```
<soap:envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <soap:body>
        <Vynásobit>      <!--Názov funkcie, ktorú má služba spustiť.-->
            <a>2.0</a> <!--Prvá vstupná hodnota odovzdávaná funkcií.-->
            <b>7</b>   <!--Druhá vstupná hodnota odovzdávaná funkcií.-->
        </Vynásobit>
    </soap:body>
</soap:envelope>
```

Účelom zasania tejto správy je snaha o spustenie funkcie s názvom „*Vynásobit*“ a pritom tejto funkcií poskytnúť dve vstupné hodnoty, t. j. dve konkrétné čísla, ktoré má táto funkcia použiť vo výpočte. Znova pripomíname, že takéto riešenie sa dá použiť iba vtedy, ak

služba, ktorá takúto správu zasiela inej službe, pozná presný názov funkcie, ktorú chce spustiť, a ak zároveň vie, aké vstupné údaje táto funkcia požaduje od toho, kto ju spúšťa.

Na rozdiel od správ vytvorených v štýle *RPC*, sa v správach vytvorených v štýle **document nešpecifikuje názov funkcie, ktorá sa má spustiť**. Predpokladá sa teda, že webová služba, ktorá takúto správu prijme, sama rozpozná, ktorú zo svojich funkcií má spustiť tak, aby vhodným spôsobom zareagovala na tento podnet. Obsah takejto správy sa ďalej odvíja od toho, či ide o štýl *document/encoded* alebo *document/literal*. Príkladom správy vytvorenej v štýle *document* je nasledujúci kód v jazyku SOAP.

```
<soap:envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <soap:body>
        <!--Údaje zasielané v tele správy-->
        <tovary>
            <tovar>
                <názov>Čokoláda Orion biela</názov>
                <ID>O32</ID>
                <počet_kusov>30</počet_kusov>
            </tovar>
            <tovar>
                <názov>Jogurt Zvolenský čučoriedkový</názov>
                <ID>Z11</ID>
                <počet_kusov>25</počet_kusov>
            </tovar>
        </tovary>
    </soap:body>
</soap:envelope>
```

Ako sme už uviedli, správy štýlu *document*, ako aj správy štýlu *RPC* môžu byť dvoch typov, ktorými sú [ANG11]:

- *Encoded* – v tomto prípade musí byť SOAP správa štruktúrovaná podľa konkrétnej XML schémy. Táto schéma pritom musí byť buď priamo súčasťou WSDL dokumentu

(v elemente *types*), ktorý opisuje cieľovú službu, alebo musí byť k nemu pripojená cez odkaz na externý súbor.

- *Literal* – SOAP správa nemusí byť štruktúrovaná podľa žiadnej konkrétnej XML schémy, no aj tak musí zodpovedať základným požiadavkám uvedeným vo WSDL dokumente.

Ak daná služba prijme vstupnú SOAP správu s určitou požiadavkou, no z nejakých príčin túto požiadavku nemôže obslužiť, zvykne do svojej výstupnej správy s odpoveďou zapísat' informácie o chybe, resp. príčine svojho zlyhania. Takéto informácie sa zvyknú uvádzat' do tzv. **chybovej časti**, ktorá je v *tele SOAP správy* reprezentovaná elementom *fault*. Do elementu *fault* bývajú vnorené ďalšie elementy, ktoré vzniknutú chybu bližšie špecifikujú. Takýmito elementmi môžu byť [W3C01]:

- *faultcode* – do tohto elementu sa zapisuje pomenovanie alebo identifikačný kód chyby. Medzi základné typy chýb patria *versionMismatch* (služba spracovávajúca prijatú správu zistila, že v elemente *envelope* tejto správy bol uvedený neplatný menný priestor), *mustUnderstand* (služba spracovávajúca prijatú správu nebola schopná spracovať niektorú časť jej hlavičky, hoci to bolo jej povinnosťou, pretože atribút *mustUnderstand* pre túto časť mal hodnotu jedna), *client* (služba spracovávajúca správu zistila, že táto správa nebola vhodne štruktúrovaná, t. j. neobsahovala potrebné údaje – k chybe teda došlo na strane klienta), *server* (služba spracovávajúca správu zistila, že nebola schopná spracovať správu, no nie kvôli chybe v tejto správe, t. j. kvôli chybe na strane klienta, ale kvôli vlastnej chybe),
- *faultstring* – do tohto elementu sa zapisuje opis chyby vo forme ľudskej reči – tento opis je teda určený pre človeka a nie pre strojové spracovanie prostredníctvom počítačového programu,
- *faultactor* – do tohto elementu sa zapisuje fyzická adresa (URI) sprostredkovateľa, ktorý na ceste správy spôsobil chybu. Používa sa len v prípade, ak chyba nastala až po prvotnom odoslani správy jej počiatočným odosielateľom (klientom),
- *detail* – používa sa na bližšiu špecifikáciu toho, k akej chybe došlo pri spracovávaní obsahu tela prijatej správy. Ak došlo k chybe pri spracovávaní hlavičky prijatej správy, potom sa element *detail* nepoužíva a podrobnosti o chybe sa namiesto toho zapisujú do hlavičky správy s odpoveďou.

Príkladom začlenenia chybového hlásenia do SOAP správy, ktorá predstavuje odpoveď na prijatú požiadavku, je tento zdrojový kód [ERL09]:

```
<soap:envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <soap:body>
        <soap:fault>
            <soap:faultcode>
                MustUnderstand
            </soap:faultcode>
            <soap:faultstring>
                Hlavička prijatej správy nebola rozpoznaná.
            </soap:faultstring>
            <soap:detail>
                <x:appMessage
                    xmlns:x="http://www.xmltc.com/tls/faults">
                    Hlavička CorrelationID nebola spracovaná
                    prijemcom, hoci sa jej spracovanie vyžaduje.
                </x:appMessage>
            </soap:detail>
        </soap:fault>
    </soap:body>
</soap:envelope>
```

## Zhrnutie kapitoly 6

SOA je založená na používaní otvorených štandardov. Otvorený štandard v IT je publikovaná (t. j. zverejnená) špecifikácia určitého IT riešenia spravidla vytvorená štandardizačnou organizáciou, ktorá netrvá na žiadnych právnych obmedzeniach pri používaní tejto špecifikácie inými firmami alebo organizáciami (t. j. môže ju používať každý, kto má o to záujem). Jazyk XML je štandard, ktorý slúži na štruktúrovaný zápis údajov (t. j. taký zápis, kde každý údaj má priradený určitý význam a údaje sú zoradené tak, že sú medzi nimi jasne viditeľné vzťahy – hierarchia údajov). Vďaka XML sa údaje dajú platformovo nezávisle uchovávať na akomkoľvek výpočtovom zariadení a dajú sa ľahko prenášať pomocou prenosových protokolov,

akými sú napr. HTTP, resp. HTTPS, FTP, SMTP a iné. Okrem toho sú na jazyku XML postavené aj ďalšie jazyky ako napr. XSD, WSDL a SOAP, ktoré dodržiavajú všetky pravidlá jazyka XML, takže každý dokument zapísaný v týchto jazykoch je zároveň aj XML dokumentom. Jazyk XSD slúži na tvorbu schém pre XML dokumenty. Tieto schémy slúžia ako šablóny, ktoré predpisujú, aké údaje sa majú vyskytovať v XML dokumente; v akom poradí a aké majú mať dátové typy. Zmysel používania takýchto šablón spočíva v kontrole toho, či XML dokument obsahuje všetky údaje, ktoré má obsahovať, a či v ňom nie sú chyby. Ak XML dokument nezodpovedá presne štruktúre, ktorá je predpísaná v XML schéme pomocou jazyka XSD, potom tento dokument nie je vzhľadom na túto schému validný, z čoho jednoznačne vyplýva, že obsahuje chyby. Jazyk WSDL je štandard, ktorý slúži na opis služby. Tento opis sa pritom nezameriava na to, ako presne služba funguje (služba sa má totiž navonok javiť ako čierna skrinka, skrývajúca algoritmy, ktoré sú v nej implementované), ale slúži na to, aby sa ostatné služby dozvedeli, ako sa môžu s danou službou spojiť. Ostatné služby teda vďaka WSDL dokumentu vedia, aké vstupné údaje majú danej službe poslať, ak chcú, aby im táto služba poskytla nejakú funkcionality; ako majú byť tieto údaje štruktúrované (t. j. aké dátové typy majú mať; v akom majú byť poradí; aký majú mať formát – napr. SOAP 1.1, SOAP 1.2, MIME a pod.); akým prenosovým protokolom ich majú poslať a na aký port. Tiež sa dozvedia, aké výstupné údaje im táto služba poskytne a v akom formáte. Jazyk SOAP predstavuje štandard na zápis správ posielaných medzi službami. Tieto správy slúžia na prenos údajov a prenos inštrukcií od služby k službe. SOAP nie je prenosový protokol (t. j. sám o sebe neslúži na prenos údajov), ale je to protokol na zápis správ a tieto správy sú potom prenášané pomocou štandardných prenosových protokolov, akými sú napr. HTTP, resp. HTTPS, TCP/IP, FTP, SMTP a iné.

## Literatúra ku kapitole 6

- [ANG11] ANGSTADT, M.: *WSDL SOAP bindings confusion – RPC vs document*. In: *mangstacular.blogspot.com* [online]. 2011. [Cit. 19. 8. 2017]. Dostupné na internete: <<http://mangstacular.blogspot.sk/2011/05/wsdl-soap-bindings-confusion-rpc-vs.html>>.
- [BUT05] BUTEK, R.: *Which style of WSDL should I use?* In: *ibm.com* [online]. 2005. [Cit. 19. 8. 2017]. Dostupné na internete: <<https://www.ibm.com/developerworks/library/ws-whichwsdl/>>.
- [CLA91] CLAUS, V. – SCHVILL, A.: *Lexikón informatiky*. Bratislava: Slovenské pedagogické nakladateľstvo, 1991. 544 s. ISBN 80-08-00755-9.