MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Knižnica pre prístup k dátovým zdrojom

BAKALÁRSKA PRÁCA

Filip Bittara

Brno, 2015

Poďakovanie

Rád by som poďakoval vedúcej svojej práce doc. RNDr. Eve Hladkej, Ph.D. za pomoc a trpezlivosť, Mgr. Antonínovi Rozsypalovi, Ph.D. za cenné rady a pripomienky a mojej rodine za podporu popri štúdiu a v živote.
Špeciálne poďakovanie venujem Bc. Dominike Lacikovej, Zdenke Lacikovej a Mgr. Daniele Grznárovej za pomoc pri korektúre práce.

Zhrnutie

Cieľom tejto práce je vytvoriť aplikáciu, ktorá pristupuje k dátovým zdrojom a môže byť nasadená a využívaná v produkčnom prostredí spoločnosti ON Semiconductor. Knižnica prístup ku zdrojom zjednodušuje a do značnej miery unifikuje.
V teoretickej časti sa práca venuje popisu architektúry Service Oriented Architecture, architektonického modelu Enterprise Service Bus a zoznamuje so systémom spoločnosti, na týchto modeloch založenom.

Kľúčové slová

SOA, Architektúra orientovaná na služby, ESB, Podniková zbernica služieb, knižnica, dátové zdroje, unifikovaný prístup, Java

Obsah

Kapitola 1
Úvod

1.1 Motivácia a ciele
Tvárou v tvár podnikateľskému prostrediu manažéri požadujú efektívne riešenia, ktoré im môžu pomôcť dosiahnuť lepšie obchodné výsledky s menšou námahou. Riešenia, ktoré spájajú znižovanie nákladov so zvyšovaním adaptability, sú kritické pre spoločnosti snažiace sa prispôsobiť svojim zákazníkom, reagovať na nepredvídané udalosti a smerovať k rastu.
Možným riešením je obrátiť sa na SOA, architektúru zameranú na služby, po anglicky *service oriented architecture*, návrhový vzor, ktorého základným princípom je využívanie služieb.
Služby sú v kontexte podnikovej architektúry mechanizmy prístupu k funkcionalite príbuzného softvéru za dodržania pravidiel špecifikovaných systémom. Typicky sú realizované použitím rozhrania s popisom týchto funkcií **[1].** So SOA môžu firmy využiť možnosti, ktoré inteligentné a vzájomne prepojené systémy ponúkajú. Z hľadiska praktického využitia nás zaujíma, že návrh aplikácie podľa princípov architektúry zameranej na služby pomáha odhaľovať jej funkcie iným aplikáciám. Tak zjednodušuje prístup a zabezpečuje lepšiu vzájomnú integráciu komponentov **[2]**.
Architektonický model ESB, podniková zbernica služieb, po anglicky *enterprise service bus*, zavádza SOA integráciou izolovaných aplikácií do decentralizovanej infraštruktúry **[3].**
Táto zodpovedá za monitorovanie a kontrolu smerovania výmeny správ medzi komponentmi a dokáže riešiť možné konflikty v ich komunikácii. Vzhľadom k potrebe komponentov spracovávať dáta pochádzajúce z externých systémov je nutné zabezpečiť ich komunikáciu so zdrojmi týchto dát.
Cieľom práce je navrhnúť a implementovať knižnicu zabezpečujúcu zjednotený a zjednodušený prístup k dátovým zdrojom v programovacom jazyku Java. Knižnica musí fungovať ako súčasť konkrétneho nasadenia SOA prostredníctvom podnikovej zbernice služieb na platforme Oracle Fusion Middleware.
Knižnica, ktorú som použil, pozostáva zo šiestich kľúčových komponentov, pričom každý z nich zabezpečuje komunikáciu s iným typom dátového zdroja. Z nichfunkcie komponentov poskytujú ostatným častiam systému dáta, ktoré sú ďalej využívané a spracovávané. Funkcie zároveň umožňujúodosielanie spracovaných dát na dátové zdroje. Rozhrania komponentov sú do určitej miery zjednotené, dôsledkom čoho je zjednodušenie prechodu medzi rôznymi zdrojmi dát. Aplikácia podporuje protokoly: FTP – protokol prenosu súborov, SMTP – jednoduchý protokol na prenos pošty, HTTP – hypertextový prenosový protokol a SOAP – protokol na jednoduchý prístup k objektom. Ďalej umožňuje komunikáciu s JMS, službou Java správ, po anglicky *Java message service*. S použitím SQL, štruktúrovaného dopytovacieho jazyka, po anglicky *structured query language,* dokáže pracovať s databázami. Ku knižnici je dostupné webové rozhranie, ktoré umožňuje jednoduché nastavenie

prístupových údajov. Súčasťou riešenia je aj štatistický modul zaznamenávajúci počet prístupov k zdrojom a množstvo prenesených dát. Tieto informácie sú prístupné aplikácii, ktorá knižnicu používa a po ukončení práce sú uložené do databázy na neskoršie spracovanie.
Vzhľadom na požiadavky spoločnosti ON Semiconductor, pre ktorú je systém vyvíjaný, je pri tvorbe braný ohľad na jeho zabezpečenie a sú urobené záťažové testy. Zdrojový kód knižnice je podrobne okomentovaný. Riešenie je navrhnuté tak, aby jeho životný cyklus mohli udržiavať súčasní vývojári spoločnosti. Prípadné dodatočné úpravy by mali byť možné bez ďalšieho školenia, teda s minimálnymi nákladmi.
V súčasnej dobe je tento komponent nasadený v produkcii a využívaný zákazníkmi spoločnosti.

1.2 Prehľad práce

Práca je rozdelená do piatich kapitol.
Prvá kapitola je venovaná uvedeniu do problematiky, stručnému vysvetleniu základných pojmov v oblasti, popisu motivácie a cieľom práce. V ďalšej časti sa kapitola zaoberá bližším prehľadom práce.
V druhej kapitole sa nachádza bližšie predstavenie architektúry zameranej na služby, jej história, popis princípov a príklady najčastejšieho použitia. Popisuje implementáciu s pomocou podnikovej zbernice služieb. Kapitola predstavuje vlastnosti architektonického modelu a jeho výhody.
Tretia kapitola oboznamuje so systémom realizovaným s využitím SOA, vo vnútri ktorého daná knižnica funguje, a ďalej sa zaoberá stručným popisom ostatných komponentov.
Obsahom štvrtej kapitoly je rozbor uvažovaných riešení a návrh vyvíjaného komponentu. Jeho funkcionalita je vysvetlená a vyobrazená za pomoci UML diagramov. Popisuje implementáciu knižnice a postup jej vytvárania, testy hotovej aplikácie v rámci celého systému, problémy, ktoré boli v priebehu testovania zistené a ich riešenie.
Posledná, piata kapitola, je záverečným zhrnutím práce. Prezentuje dosiahnuté výsledky a rozoberá možnosti ďalšieho vývoja.

Kapitola 2
Dizajnový vzor SOA

2.1 Princípy
Softvérové architektúry majú niektoré špecifické vlastnosti a princípy, ktoré musia byť dodržané, aby ich funkcie boli plne využité **[4].**
Architektúra orientovaná na služby má nasledovné vlastnosti **[4, 5]**:

2.1.1 Služby sú zistiteľné a dynamicky viazané
SOA poskytuje spôsob, ako objavovať služby dynamicky za behu. Spotrebiteľ požiada registre o požadovanú službu a v odpovedi dostane potrebné informácie na vykonanie služby. Podobne rozhrania sú objavované dynamicky a správy sú taktiež stavané za behu, takže kompilácia nie je viazaná na službu. Jediná vec, ktorú je potrebné aktualizovať, je konfigurácia služby, ak sa mení.

2.1.2 Služby sú sebestačné a modulárne
Služba má súbor funkčných rozhraní, ktoré spoločne vykonávajú úlohy. Tieto rozhrania majú väzby na vytvorenie modulu a majú dostatok informácií k funkcii bez toho, aby boli autentifikované či závislé na inej softvérovej aplikácii, alebo module.

2.1.3 Služby vzájomne spolupracujú
Jedným z hlavných rysov servisne orientovanej architektúry je interoperabilita, čo je schopnosť komunikovať s rôznymi systémami bez závislosti na konkrétnej platforme alebo jazyku. Každý modul alebo softvérová architektúra má nejakú proprietárnu štruktúru, ktorá je obmedzená na svoju vlastnú doménu a je považovaná za pevne spojenú. SOA dosahuje interoperabilitu prostredníctvom podpory protokolov a dátových formátov od súčasných a od potenciálnych spotrebiteľov služby.

2.1.4 Služby sú voľne spojené
Združovanie sa vzťahuje na úroveň závislostí medzi softvérovými modulmi. Väzby môžu byť kvalifikované ako buď voľne viazané alebo pevne prepojené. Voľne viazané moduly majú dobre definované závislosti a sú flexibilnejšie. Väčšina softvérových architektúr sa snaží poskytnúť moduly, ktoré sú tak voľne viazané, ako je to možné. Naopak pevne spojené softvérové systémy sú náročné na organizáciu, pretože majú neznáme požiadavky pre každý modul v rámci štruktúry alebo architektúry softvéru. Architektúra orientovaná na služby kladie dôraz na rozvoj voľne previazaných služieb, čo znamená, že závislosti medzi spotrebiteľmi a poskytovateľmi by mali byť minimálne. V SOA sa voľné spojenie vykonáva použitím registra, ktorý obsahuje väzby a informácie o každej službe. Tesné spojenie sa pri implementácii realizuje definíciou rozhrania alebo väzbou na konkrétny protokol.

2.1.5 Služby majú rozhrania adresovateľné v sieti

Služba by mala mať adresu siete, kde je jeho rozhranie publikované. To je jeden z hlavných dizajnových princípov servisne orientovanej architektúry. Spotrebiteľ by mal byť schopný vyvolať službu v sieti distribuovaným spôsobom, čo robí služby znovu použiteľné a k dispozícii rôznym spotrebiteľom. Služby môžu byť tiež spúšťané prostredníctvom lokálneho rozhrania bez zapojenia siete. To je možné, keď sú obaja, poskytovateľ aj spotrebiteľ, na rovnakom stroji. Toto je využívané najmä pre zlepšenie výkonnosti.

2.1.6 Služby majú hrubo granulované rozhrania

Pojem granularity s ohľadom na služby môže byť definovaný ako rozsah implementácie rozhrania pre každú metódu v rámci služby. Zjednodušene, môže byť videný ako implementácia rozhrania v systéme. Existujú dve úrovne zrnitosti rozhrania – v prípade, ak rozhranie poskytuje všetky biznis funkcie, je považované za hrubozrnné. Na druhej strane, jemnozrnné rozhranie implementuje len časť biznis funkcionality. Servisne orientovaný systém štandardne podporuje hrubozrnné vonkajšie rozhranie s rôznou zrnitosťou pre rozhrania medzi službami. To znamená, že objekty v rámci služby môžu byť jemnozrnné, ale iba ak tieto objekty zostanú v rámci fyzickej štruktúry služby.

2.1.7 Služby majú transparentnú lokalitu

Transparentnosť umiestnenia je jednou z hlavných charakteristík, ktoré SOA podporuje. Priehľadnosť umiestnenia umožňuje služby presunúť z jedného miesta na druhé bez toho, aby to ovplyvnilo spotrebiteľa služby. Spotrebiteľ nevie o umiestnení služby, kým sa nepozrie na jej beh v registri. Spotrebiteľ je závislý len na službe, čo uľahčuje zmenu umiestnenia implementácie služby.

2.1.8 Služby môžu by skladané do nových aplikácií

Znovupoužiteľnosť služieb je tiež jednou z kľúčových charakteristík SOA. To umožňuje rozvoj nových aplikácií využívajúcich existujúce služby. Zloženie služieb je efektívny prístup k dizajnu, ktorý sa zameriava na znovupoužiteľnosť funkcionality služieb bez nutnosti akejkoľvek znalosti o jeho budúcom využití.

2.1.9 SOA podporuje samoliečenie

Samoliečebný systém má schopnosť obnoviť sa do normálneho stavu, keď dôjde k chybe. Aplikácie v rámci systému na báze SOA by sa mali dokázať takto obnoviť. Systém založený na SOA by mal pre efektívne a správne fungovanie vždy podporovať samoliečebné vlastnosti.

2.2 História

Termín SOA, architektúra orientovaná na služby, bol po prvýkrát použitý v roku 1994 Alexandrom Pasikom, bývalým analytikom spoločnosti Gartner, počas kurzu o middleware architektúre, ktorý v tom čase vyučoval.

Tento termín zaviedol, pretože rozdelenie architektúry na klient a server začalo strácať svoj pôvodný význam. Mnoho ľudí začalo používať termín „klient/server" ako označenie pre distribuovaný výpočet medzi osobným a iným počítačom. V tomto poňatí je klient počítačom spracúvajúcim prezentačnú logiku a väčšinu biznis logiky a server obsluhuje databázu, ukladá dáta a prípadne spracúva časť biznis logiky. Výrazy „klient" a „server" typicky označujú hardvér.

Aby predišiel problémom, ktoré vznikali kvôli rozdielom medzi pôvodným a novým významom „klient/server", Pasik zaviedol nový termín, pričom zdôrazňoval potrebu orientácie na serverovú časť a povzbudzoval vývojárov k návrhu SOA aplikácií **[6].**

Pre lepšie porozumenie konceptu architektúry orientovanej na služby je nutné pochopiť, že nevznikla ako samostatný koncept, ale bola skôr evolúciou rôznych paradigiem a technológií. Keďže SOA je iba ďalším krokom tejto evolúcie, nie je jednoduché ju presne vymedziť. Nasledujúca časť sa zameriava na technológie, z ktorých sa SOA vyvinula.

2.2.1 Objektová orientácia

V 90. rokoch 20. storočia prijala IT komunita filozofiu definujúcu cestu, ktorou by sa vývoj distribuovaných riešení mal uberať. Bola ňou objektová orientácia. Táto paradigma zaviedla princípy, ktorých dodržiavanie prispieva k udržaniu konzistencie naprieč prostrediami. Princípy definujú špecifické vzťahy medzi logickými jednotkami – objektmi, čo zabezpečuje predvídateľný priebeh vykonávania daného riešenia.

SOA je právom porovnávaná s objektovou orientáciou a môže byť považovaná za jej určitú nadstavbu, keďže mnohé princípy, napr. znovupoužiteľnosť, abstrakcia a spájanie sú pre ne spoločné. Architektúra orientovaná na služby však pridáva i niektoré ďalšie princípy **[7].**

2.2.2 Webové služby

Napriek tomu, že SOA a Webové služby sú implementáciou rozdielne, ich vzájomná previazanosť je bežná – a to až do tej miery, že hlavní poskytovatelia SOA riešení postavili svoje platformy na využívaní týchto služieb.

Dôsledkom toho je aj ovplyvnenie Web service frameworkov a prienik princípov servisnej orientácie do ich vnútra **[7, 8].**

2.2.3 Správa biznis procesov

Podnikové procesy, ktoré bežia hladko a v správnom poradí, sú pre podnik nevyhnutné. Správa podnikových procesov kladie dôraz na zefektívnenie podnikových procesov tým, že sa prispôsobuje a zlepšuje ich účinnosť.

Vrstva spracovania biznis logiky je hlavná vrstva v servisne orientovanej architektúre. Poskytuje možnosť kompozície servisov a ich vykonávanie.
Jedným z hlavných cieľov SOA je automatizácia obchodných procesov, ktoré sú vysoko adaptívneku zmene. Tento cieľ môže byť dosiahnutý presunutím obchodnej logiky do jednej vrstvy a donútením služieb znovu použiť túto logiku bez toho, aby bola znovu vytváraná. Znovupoužiteľnosť služieb je kľúčovým faktorom v rámci architektúry orientovanej na služby. Riadenie podnikových procesov plne využíva túto funkcionalitu **[7].**

2.2.4 Integrácia podnikových aplikácií
Integrácia sa stala bodom záujmu v neskorých 90. rokoch. Aplikácie v mnohých organizáciách neboli stavané na zdieľanie dát mimo hraníc systému. Ak sa objavila požiadavka na zdieľanie dát ako výsledok integrácie, často boli vytvorené point-to-point kanály. Toto viedlo k problémom spojeným s nestabilitou, nemožnosťou rozšíriteľnosti a nedostatočnou interoperabilitou.
Ako riešenie boli predstavené Enterprise Application Integration (EAI) platformy. Tieto paradigmy využívali adaptér, broker komponentu a spracúvajúce komponenty, aby dosiahli integráciu aplikácií a spracovávanie podnikových dát.
Najväčším problémom EAI boli vysoké náklady na škálovateľnosť a produkty uzamknuté na jedného predajcu. Podniky následne začali hľadať riešenie, kde by naopak bola ponuka rozmanitá a náklady na škálovateľnosť nízke. Vznikom webových služieb ako otvoreného frameworku sa podniky pohli smerom k orientácii na služby, ktorá im dala oveľa väčšiu flexibilitu a kontrolu nad ich integračnou architektúrou a platformou **[7].**

2.3 Implementácia

2.3.1 Služby
Služba je softvérový komponent popísaný meta dátami, ktorý môže byť chápaný programom **[9].** Existujú rôzne spôsoby, ako definovať služby, pretože rozdielne služby majú rozdielne atribúty. V biznis kontexte sa služby môžu značne líšiť v závislosti na požiadavkách rôznych podnikateľských subjektov, pre ktoré poskytujú funkcionalitu. Služby môžu byť použité na čítanie dát, ich zapisovanie alebo ako kompozície iných služieb, ktoré vykonávajú nejaký pracovný postup alebo proces. Z tohto dôvodu nie je možné považovať všetky služby za rovnaké. Lepším riešením je ich kategorizácia podľa atribútov a funkcií.
Služby je možné rozdeliť do nasledujúcich troch kategórií **[6]**:

- Základné služby sú najjednoduchšie služby, teda poskytujú iba základné obchodné funkcie. Inými slovami, poskytujú iba službu čítania alebo zápisu jednému konkrétnemu backendu. Ich životnosť je krátka a zvyčajne sú bezstavové. Základné služby sú ďalej rozdelené do dvoch typov:
    - Základné dátové služby – služby spojené s konkrétnym backendom a vykonávajú len čítanie a zápis. Navyše musia spĺňať ACID (atomicita, konzistentnosť, izolácia, trvanlivosť) kritériá **[10]**.

    - Základné logické služby - sú podobné základným dátovým službám. Rozdiel je v tom, že tieto služby obsahujú iba základné biznis pravidlá a logiku.
- Zložené služby sú zložením buď základných služieb, alebo iných zložených služieb. Zložené služby v kontexte SOA sú tiež označované ako orchestrácie. Sú na vyššej úrovni ako základné služby, napriek tomu, že ich životnosť je tiež krátka a väčšinou sú taktiež bezstavové. Zvyčajne bežia v biznis procese.
- Spracúvajúce služby zvyčajne predstavujú pracovný postup alebo obchodný proces. Jedná sa o činnosti/služby, ktoré bežia dlhodobo v rámci jedného pracovného postupu alebo procesu. Keďže sa jedná o dlhodobé prevádzkové služby, potrebujú udržiavať stav (napríklad nákupný košík). Tieto stavy môže byť nutné udržiavať vo viacerých reláciách. Pomáhajú pri obnove služieb pri zlyhaní, uvedením do rovnakého stavu, v akom služba bola predtým, ako k zlyhaniu došlo.

V SOA služby pracujú zvyčajne v troch rolách, ktoré prichádzajú do úvahy: žiadateľ, poskytovateľ a broker.

- Poskytovateľ služby je subjekt, ktorý má prístup k iným službám a je tiež zodpovedný za vytvorenie služby a ich publikovanie brokerovi.
- Žiadateľ o služby je osoba, ktorá vyžaduje určitú prácu vykonanú inou službou a pracuje na základe prehľadávania popisov služieb u brokera. Žiadateľ o služby je tiež zodpovedný za previazanie služieb po ich objavení.
- Broker obsahuje všetky informácie o službách, ktoré sú registrované a je tiež zodpovedný za presmerovanie požiadaviek žiadateľa na príslušného poskytovateľa.

2.3.2 Enterprise Service Bus

Podniková zbernica služieb (ESB) je softvérový architektonický konštrukt, ktorý poskytuje základné služby pre komplexné architektúry cez udalosťami riadený a na štandardoch založený systém na zasielanie správ (zbernicu) **[3]**. ESB vzor definuje model pre integráciu podnikových aplikácií pomocou obyčajnej zbernice na zasielanie správ.

Podniková zbernica služieb vychádza z Integrácie podnikových aplikácií (EAI), robí ju flexibilnejšou, uľahčuje integráciu postavenú na štandardoch a neobmedzuje ju integráciou na báze webových služieb – komunikácia nie je obmedzená len na HTTP protokol. Jednou z užitočných funkcií ESB je, že spotrebiteľ môže poslať jeho správu pomocou HTTP protokolu a poskytovateľ môže reagovať pomocou JMS a nemusia sa starať o akékoľvek prevody alebo komunikačné protokoly. ESB sa bude starať o všetky tieto problémy v oblasti infraštruktúry. Stručne povedané, ESB poskytuje dopravu správ podobne ako EAI, okrem toho však tiež poskytuje služby registra, ktorýumožňuje spotrebiteľom zistiť, čo konkrétne služby robia . Sú tiež k dispozícii rôzne mechanizmy pre dohody no úrovni služieb (SLA) ako zmluvy, zabezpečenie, smerovanie správ a transformácie správ. Tieto možnosti sa u jednotlivých výrobcov líšia **[11]**.

ESB vzor obsahuje mechanizmus pre riadenie prístupu a správu identít, mechanizmus pre pomenovanie servisov, register servisov a spoločnú zbernicu pre posielanie správ, ktorá je zodpovedná za komunikáciu medzi rôznymi protokolmi.
ESB zabezpečuje komunikáciu medzi službami registrovanými na zbernici a aplikáciami používajúcimi tieto služby bez znalosti ich fyzického umiestnenia.
Pomáha tiež pri štandardizácii správy identít, konvencií pomenovaní, formátov správ a komunikačných protokolov.
Najlepším riešením, ktoré ESB poskytuje, je skutočnosť, že akonáhle sa služba zapíše na zbernicu, môže sa na všetky ostatné servisy zapísané na zbernici pripojiť a komunikovať s nimi, čo vedie k veľmi voľne spojenému systému **[12]**.
ESB bol navrhnutý na prekonanie problémov EAI. Hlavné problémy ako point-to-point pripojenie a jediný bod zlyhania boli zbernicou podnikových služieb vyriešené.
Enterprise Service Bus môže byť považovaný za chrbticu servisne orientovanej architektúry.
Funkcie poskytované ESB sú nevyhnutné pre implementáciu SOA. Existuje mnoho predajcov, ktorí poskytujú open source a proprietárne ESB pre rôzne technológie.
Nižšie sú uvedené niektoré z možností a funkcií, ktoré podniková zbernica služieb podporuje **[13]**:

- Komunikácia – komunikácia a smerovanie sú dva najdôležitejšie prvky. Podporuje smerovanie a zasielanie správ rôznych štýlov ako napríklad požiadavka/odpoveď, publikovanie/odber a pre komunikáciu podporuje rôzne transportné protokoly, ktoré sú široko dostupné. To pomáha priehľadnosti umiestnenia, čo vedie k voľnému spojeniu medzi spotrebiteľom a službou.
- Integrácia – pre integráciu ESB poskytuje niekoľko adaptérov a integračné techniky. Tieto techniky pomáhajú pri vytváraní služieb, ktoré skrývajú technické detaily iných služieb a vystavujú len relevantné informácie pre spotrebiteľa.
- Interakcie služieb – formáty rozhraní a modely správ sú veľmi dôležité pre interoperabilnú komunikáciu medzi aplikáciami a službami. ESB podporuje rôzne modely založené na WSDL a SOAP, ktoré štandardizujú rozhranie a umožňujú službám komunikovať len detaily určené kontraktom.
- Management – ESB poskytuje tiež správu zariadení, monitorovanie a smerovanie, adresovanie, transformácie a kontrolu nad pomenovaním schopností. To pomáha pri riadení služieb v rámci podniku.

Ďalšie funkcie v Enterprise Service Bus:

- Komponenty pre spracovanie orchestrácií môžu byť pridané do ESB pre podporu krátkodobých a dlhotrvajúcich procesov.
- Sleduje a riadi kvalitu služieb.
- Poskytuje prezentačné služby, aby pomohla vytvoriť portály, ktoré kombinujú služby z rôznych zdrojov.

2.3.3 Registre

Implementačné artefakty, ktoré sú riadené zo SOA, by mali byť registrované v registri, aby sa maximalizovalo opätovné využitie a poskytla správa podnikových aktív **[15]**.

Hlavným cieľom registrov je registrovať služby a vybaviť ich schopnosťou byť dynamicky objavené. Služba je registrovaná v rámci registra tým, že poskytuje svoju adresu a meta informácie.

Konečným cieľom registra je umožniť rôznym aplikáciám spoľahlivo komunikovať, a to bez akéhokoľvek problému s interoperabilitou a s minimálnym zásahom človeka. Inými slovami, register SOA môže byť videný ako zdroj, ktorý poskytuje riadený prístup k podstatným údajom pre správu systémov založených na servisne orientovanej architektúre. Je to katalóg, ktorý obsahuje informácie o dostupných službách. Tento katalóg je neustále aktualizovaný a informácie o službách sú pridané, aktualizované ačasto mazané. Register SOA umožňuje spotrebiteľovi, aby dynamicky objavil služby, ktoré chce použiť.

Register SOA môže byť implementovaný pomocou rôznych prístupov. Najbežnejší prístup je UDDI (Univerzálny popis, objavenie a integrácia), špecifikácia, ktorá je založená na XML registri. Bola vyvinutá za účelom výroby systémov interoperabilných pre e-komerciu a interakciu medzi webovými službami.

Register SOA stavia na UDDI tým, že pridá funkciu kontinuálnej revízie obsahu. Tento druh obsahu môže existovať vo forme XML dokumentov, pracovných postupov alebo procesov.

SOA stavia na UDDI a uľahčuje podnikovým aplikáciám objavovať a využívať aktuálne a relevantné informácie, a to účinnejším spôsobom než pomocou UDDI **[15, 16]**.

2.3.4 Portály

Portály sú voliteľné add-ony pre servisne orientovanú architektúru. Sú používané primárne s cieľom vytvoriť správcovskú konzolu alebo integrovať front-end aplikácie do paradigmy SOA. Pri každom návrhu softvéru sú najprv vyvinuté rozhrania, aby bolo možné vytvoriť si obraz finálneho produktu – tým môže byť desktopová alebo webová aplikácia. Podobne, portály pri SOA tiež môžu pôsobiť ako prvý logický krok v procese návrhu **[14]**.

Prvým krokom SOA je definovať požadované služby a aplikácie, ktoré budú využívať tieto služby. Portály zvyčajne vykonávajú rovnakú činnosť identifikácie potrebných služieb a aplikácií, ktoré budú tieto služby využívať. Portály teda môžu byť považované za prvý krok k implementácii SOA.

V koncepte SOA môžu portály zohrávať dôležitú úlohu pri prijímaní rozhodnutí o tom, aká technológia by mala byť použitá na vykonanie požadovaných funkcií. Stručne povedané, portály poskytujú užívateľské rozhranie pre interakciu so službami – ich návrh a dôležitosť závisí na poskytovateľovi služieb a spotrebiteľovi. Portály sú umiestnené v najvyššej vrstve SOA. ESB je prístupná pre všetky tieto vrstvy.

Kapitola 3
Prostredie Unified Environment

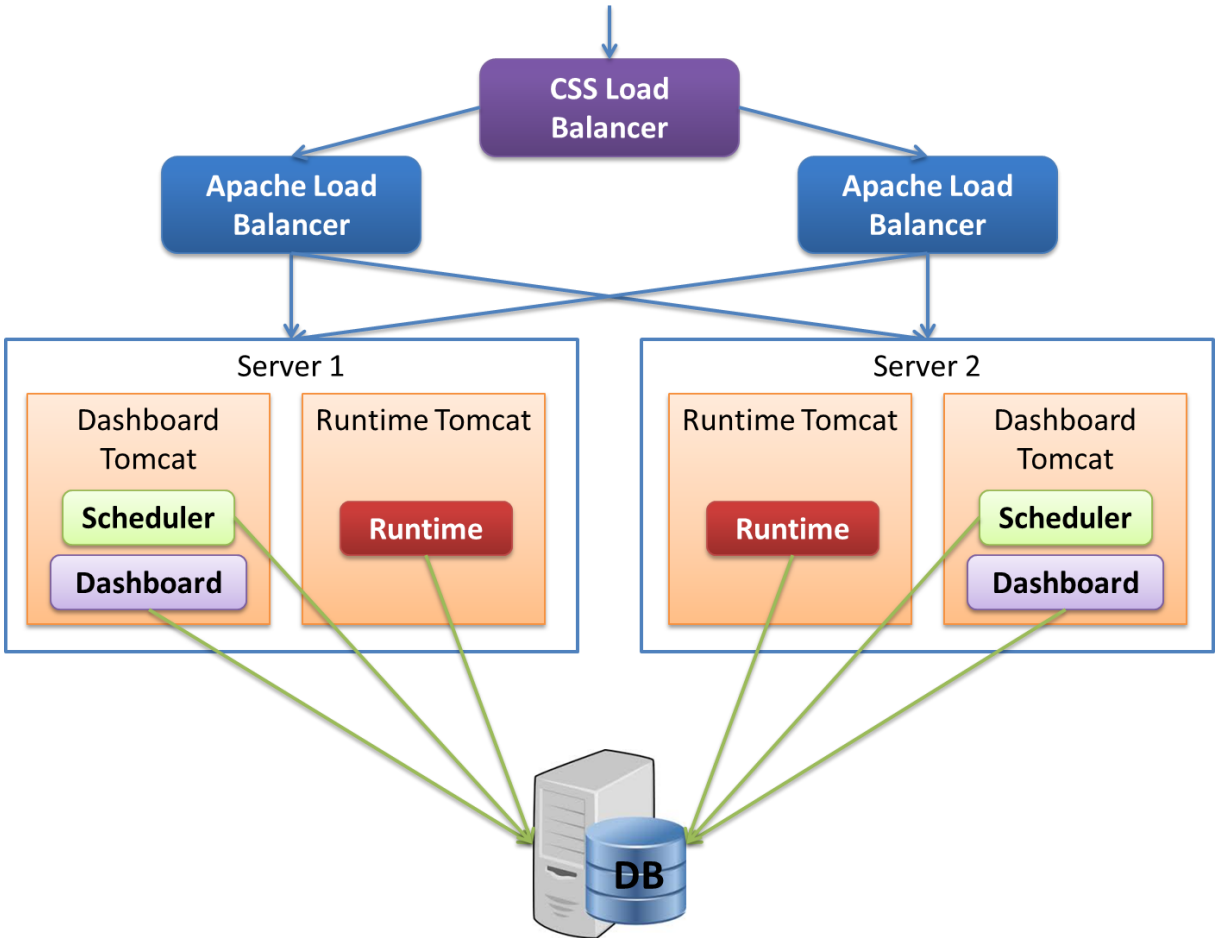
3.1 Predstavenie systému
Unified Environment je sada aplikácií bežiacich v J2EE kontajneri Tomcat s databázou Oracle v pozadí. Hlavným účelom prostredia je úplné nahradenie systému postaveného na JCAPS5 a JCAPS6, a zároveň poskytnutie pridanej hodnoty zavedením novej funkcionality. Celé riešenie je na mieru vyvinuté v jazyku Java za využitia technológií Spring, Quartz, JGroups, JQuery a ďalších open source technológií a frameworkov.
Hlavné bodynávrhu a vývoja riešenia sú nasledovné:
vytvoriť prostredie, ktoré je schopné spustiť existujúce JCAPS5 a JCAPS6 rozhrania s minimálnymi zmenami, poskytnúť vysoko dostupné prostredie s minimálnymi prestojmi, presunúť manažment rozhraní smerom k aplikačným tímom sprístupnením záznamov a konfigurácie majiteľom a poskytnúť im všetky potrebné nástroje a školenia, povoliť aplikačným tímom podieľať sa na vývoji rozhraní a poskytnúť nástroje pre vytvorenie jednoduchých riešení s nastavením iba základných údajov,
zamerať sa na sledovanie aplikácií, oznamovanie a hlásenie pre ľahkú systémovú podporu a údržbu.

3.2 Technický prehľad
3.2.1 Architektúra
Celé prostredie je navrhnuté tak, aby bolo vždy k dispozícii – pozostáva z viacerých uzlov bežiacich v klastri. Architektúra klastra je znázornená na obrázku 1 – architektúra prostredia Unified Environment.
Hlavné zložky prostredia Unified Environment sú nasledovné:

- Runtime engine – prostredie, v ktorom sú všetky rozhrania spúšťané.
- Dashboard – front-end GUI pre prezeranie a úpravu konfigurácie rozhraní, prezeranie záznamov, monitorovanie prostredia a podávanie správ. Udržiava dokumentáciu jednotlivých rozhraní. Je navrhnutá tak, aby bola jediným bodom pre všetky informácie týkajúce sa rozhraní a celého prostredia.
- Scheduler – Komponent pre spoľahlivé plánovanie spúšťania rozhraní. J2EE aplikácia postavená na základe frameworku Quartz.

Vstupným bodom je IP adresa vyrovnávača zaťaženia CSS – riešenie postavené na hardvérovom vyrovnávači zaťaženia od spoločnosti Cisco. CSS potom smeruje požiadavku na vyrovnávače záťaže Apache využívajúc metódu Round Robina sleduje ich dostupnosť. Round Robin je metóda, pri ktorej požiadavkuvykonáva RR po určitý pevne stanovený časový rámecnamiesto jej vykonávania až do jej skončenia **[17]**. V prípade, že niektorý z vyrovnávačov prestal fungovať, je všetka prevádzka

smerovaná na zostávajúce Apache vyrovnávače. Každý z nich obsahuje sadu mapovaní pre jednotlivé zložky.
Dashboard je balancovaný v režime sticky session - tým je zaistené, že každý užívateľ je vždy smerovaný na rovnaký uzol počas celého trvania jeho relácie. Iba v prípade, že cieľový uzol nie je k dispozícii, je užívateľ presmerovaný k jednému z ostatných uzlov.
Zaťaženie Runtime Engine je vyrovnávané v režime Round Robin. Neexistuje žiadna relácia, ktorá by mala byť zachovaná a tento prístup zabezpečuje rovnomernejšie rozdelenie zaťaženia.



Obrázok 1 – architektúra prostredia Unified Environment

3.2.2 Použité technológie
Celé riešenie je postavené pomocou programovacieho jazyka Java, J2EE a open source technológií. Nasleduje zoznam použitých technológií a ich využitie:

- Java 7 – celé prostredie je vyvinuté v najnovšej verzii jazyka Java v období jeho implementácie.
- Oracle – celé prostredie(konfigurácie, záznamy) je uložené v Oracle 11i databáze. Databáza je jedinou zložkou, ktorá nie je v klastri, a tak sa stáva

potenciálnym jediným bodom zlyhania. Dôvodom sú cenovo neprístupné licencie databázových a klastrových riešení od spoločnosti Oracle.

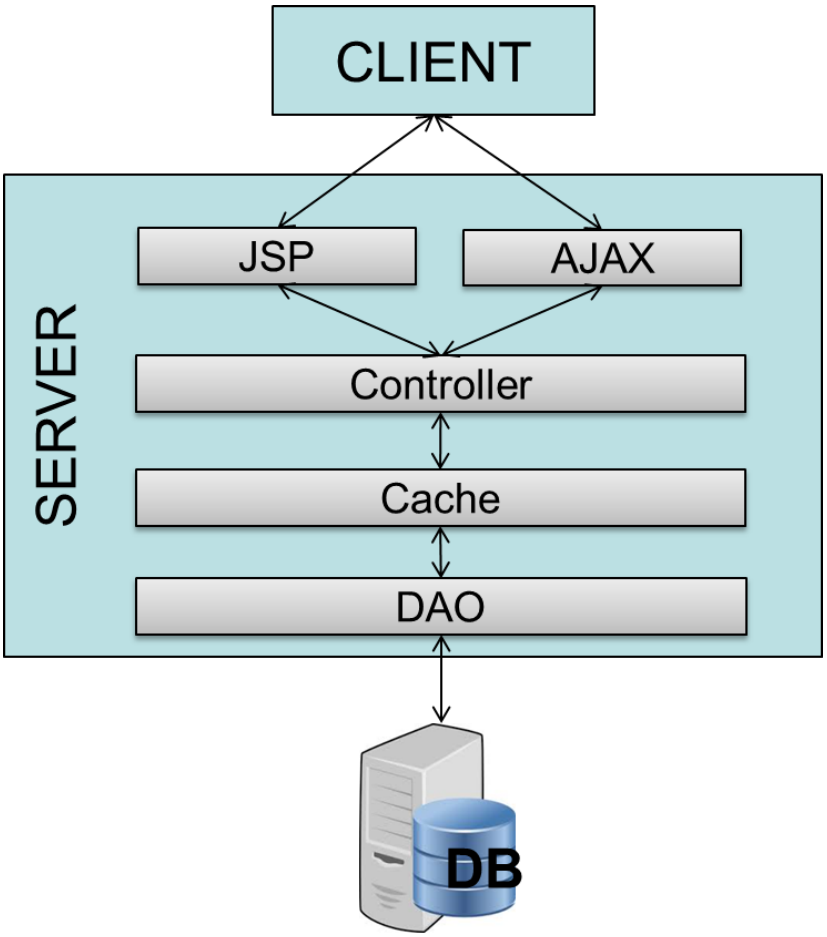
- Tomcat 7 – aplikačný server je hostiteľom prostredia. Tomcat je open source server bez plnej podpory J2EE. Je dôležité, aby všetky komponenty, ktoré sú potrebné pre beh aplikácie, boli zahrnuté v jej vnútri. Táto skutočnosť robí server nenáročným a rýchlym. Navyše jeho vysoká miera nasadenia a veľká komunita ho robia správnou voľbou pre riešenia, kde nie je požadovaná plná podpora J2EE.
- Spring – hojne využívaný naprieč celým prostredím. Spring je veľký a stále rastúci framework s radou modulov. V riešení sú použité nasledujúce moduly:
    - Configuration - všetky komponenty sú definované v konfiguračnom súbore Springu a používajú paradigmu Inversion of Control.
    - Database access – využitie Spring JDBC šablóny napomáha ľahkému prístupu k databáze. Vytvára obal okolo štandardnej JDBC, starajúcej sa o často využívaný kód a poskytuje vyššiu úroveň API.
    - MVC – návrhový vzor pre webové aplikácie. Existuje veľa populárnych frameworkov pre implementáciu MVC – Spring MVC je jedným z nich. Pretože Spring sa používa v tomto riešení pomerne často, bola to prirodzená voľba. Modul MVC zaisťuje dobre štruktúrovaný dizajn oddelením aplikačnej logiky a zobrazenia. Ďalšou výhodou je, že modul MVC poskytuje vyššiu úroveň API pre bežné úlohy, ako napríklad mapovanie parametrov a mapovanie adries URL.
- AJAX – technológie pre načítanie iba časti webovej stránky. Aplikácie sú tak responzívnejšie a užívateľsky prívetivé. Svojím spôsobom sa aplikácia priblíži klasickým desktopovým aplikáciám pri zachovaní všetkých výhod aplikácií bežiacich na serveri.
- JQuery – JavaScript framework, ktorý zjednocuje API pre rôzne typy prehliadačov a poskytuje vyššiu úroveň API.
- JGroups – open source framework pre zasielanie správ v rámci klastra. Rámec sa stará o správu klastra ako prvotné vytváranie, udržiavanie v chode a pridávanie nových uzlov. Takisto poskytuje niektoré komponenty zabezpečujúce posielanie a prijímanie synchrónnych správ, distribuované mapy a vyrovnávacie pamäte. Pre potreby riešenia je nad JGrups vyvinutá nadstavba pre ukladanie do vyrovnávacej pamäte.

3.2.3 MVC

Obrázok 2 – MVC Architektúra znázorňuje vrstvy použité v riešení v priebehu spracovania každej žiadosti. Každá žiadosť môže byť buď plným načítaním alebo žiadosťou o zmenu časti stránky (AJAX). Požiadavka je najprv poslaná ku kontroléru, ktorý načíta dáta z vyrovnávacej pamäte a vráti späť na zobrazenie (JSP, JSON). Aktualizácie dát znovu používajú vyrovnávaciu pamäť, ale dáta sú tiež prenášané do databázy prostredníctvom DAO a synchronizované s ostatnými inštanciami vyrovnávacej pamäte.

Nasleduje zoznam vrstiev a ich popis:

- DAO – na mieru postavené objekty na prístup k dátam – Data Access Objects (DAO). Založené na Spring JDBC šablóne a zaisťujú funkcie CRUD (vytvárať, čítať, aktualizovať, mazať).
- Vyrovnávacia pamäť – na mieru postavená replikovaná vyrovnávacia pamäť.
- Vyvinutá nad frameworkomJGroups. Každá vyrovnávacia pamäť udržuje kópiu konkrétnych dát objektu jedného typu ako napríklad rozhrania a oznámenia. Vyrovnávacia pamäť je súčasťou klastra a každá zmena je automaticky rozšírená do ostatných inštancií.
- Kontrolér – komponent tak ako je definovaný MVC návrhovým vzorom. Je to vstupný bod pre každú požiadavku a je zodpovedný za jeho odbavenie. Typicky volá logiku (Cache, DAO), robí drobné transformácie dát a poskytuje dáta komponentu zobrazenia.
- Zobrazenie – prezentačná vrstva, ktorá generuje pohľad vrátený späť klientovi. Formát je typicky HTML kódzaloženýna JSP alebo dáta JSON použitépre widgety na strane klienta, ako je JQGrid.
- Klient– Webový prehliadač, ktorý spustí aplikáciu na počítači používateľa. Aplikácia nie je čistým HTML, namiesto toho je bohato zmiešaná s JavaScriptom. Obsah je načítaný jadrom aplikácie ako HTML stránky s nejakým Javascriptom. Javascript potom stará o spracovanie udalostí alebo načítanie ďalšej časti stránky. Javascript často definuje widget ako napríklad JQGrid tabuľku, widget sa pripojí k serveru a načíta dáta a dáta sú nakoniec vykreslené na strane klienta prostredníctvom widgetu. Tento prístup eliminuje prepravu veľkých objemov dát – transportované sú iba potrebné dáta. Ďalšou výhodou je, že tento prístup jasne oddeľuje dáta od ich vizuálnej formy, čo umožňuje znázornenie rovnakých dát.

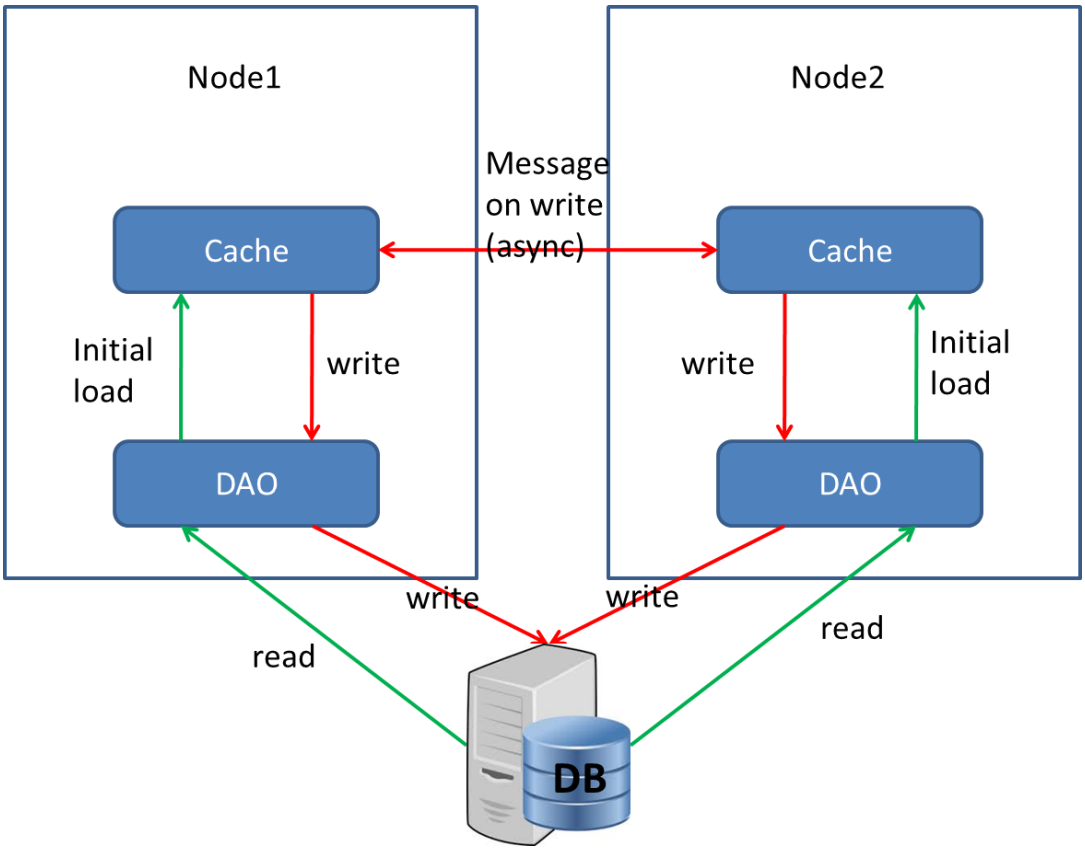


Obrázok 2 – MVC

3.2.4 Vyrovnávacia pamäť

Celé prostredie používa vyrovnávaciu pamäť na optimalizáciu spracovania požiadavky. Vyrovnávacia pamäť sa často používa v komponentoch Dashboard a Scheduler, zatiaľ čo Runtime používa cache iba ako riešenie pre zálohovanie – dáta sú načítané priamo z databázy a iba vtedy, ak zlyhá prístup k databáze, sú dáta načítané z cache. Konštrukcia aplikácií a zapadanie vyrovnávacej pamäte do konceptu sú znázornené na obrázku 3 - Vyrovnávacia pamäť.

Pri spustení aplikácie sú všetky nastavenia načítané cez DAO do pamäte. Každý typ objektu má svoj vlastný pár DAO/vyrovnávacia pamäť. Tieto pamäte sú postavené na vrchole rámca JGroups a tvoria klaster. Všetky operácie čítania sú vykonávané na miestnej inštancii vyrovnávacej pamäte. Operácie zápisu sú najprv ukladané do databázy prostredníctvom DAO, a potom zapísané do miestnej inštancie vyrovnávacej pamäte. Je zaistené, že aktualizované dáta sa šíria do všetkých ostatných pamätí v klastri.



Obrázok 3 – Vyrovnávacia pamäť

3.3 Runtime Engine

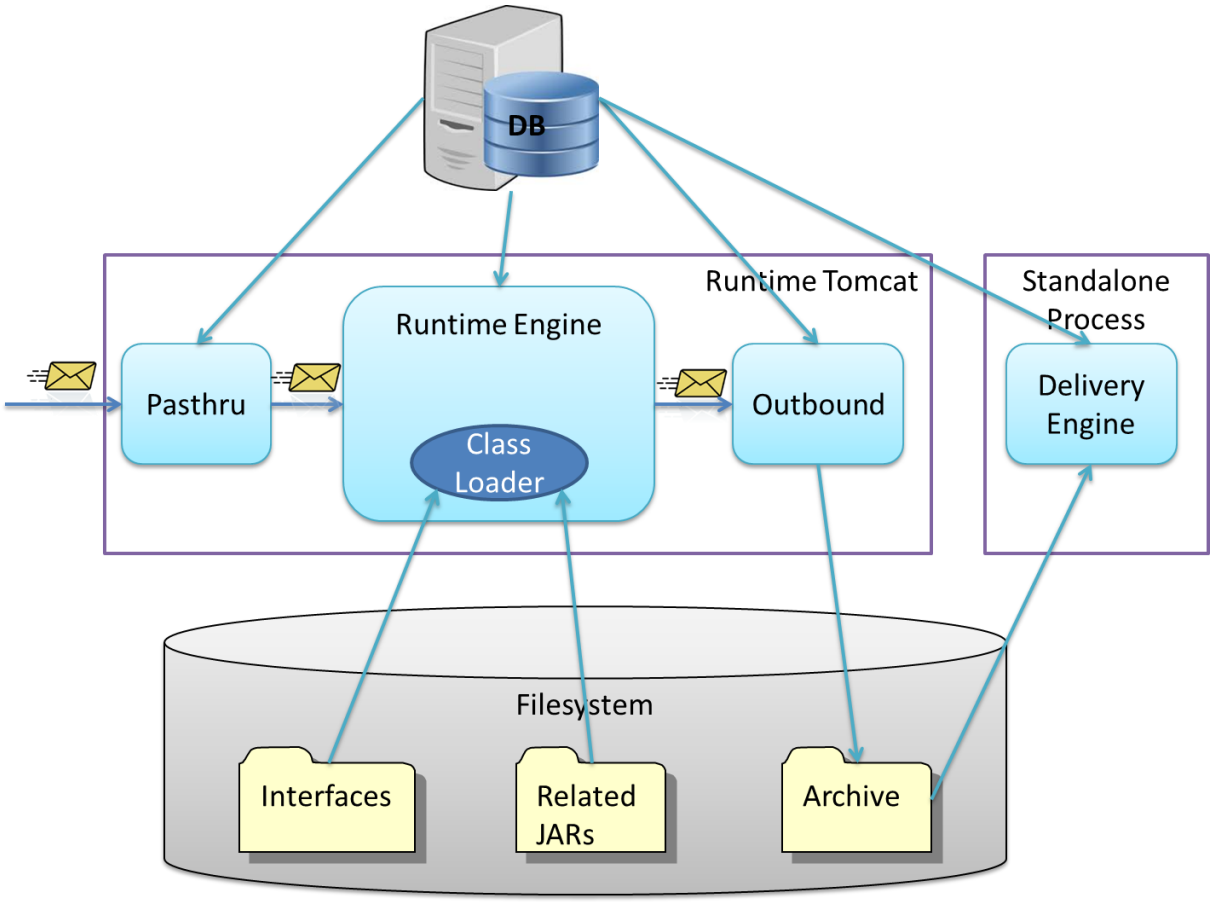
Runtime Engine je jadrom celého prostredia. Jeho úlohou je poskytnúť prostredie pre hosťovanie rozhraní a ich spúšťanie. Celý postup je znázornený na obrázku 4 – RuntimeEngine.

Každý cyklus rozhrania sa spúšťa cez spúšťaciu správu vo formáte XML. Vzhľadom k tomu, že runtime engine beží v klastri, je primárnou úlohou zistiť, či beh rovnakého rozhrania už nie je spustený na inom uzle. Runtime engine využíva JGroups replikovanú mapu pre zisťovanie, na ktorom uzle je spustený určitý beh rozhrania. Druhým krokom je kontrola zaťaženia (CPU, pamäť, počet vlákien) na aktuálnom uzle. Ak zaťaženie prekročí určitú hranicu, záťaž je vypočítaná pre všetky dostupné uzly. Žiadosť sa následne postúpi k uzlu s najnižším zaťažením.

Spúšťacia správa obsahuje meno rozhrania, prípadne adresu URL vstupného súboru a ľubovoľný počet parametrov rozhrania. Správa je odovzdaná komponentu Pasthru, ktorá stiahne vstupný súbor (ak je nakonfigurovaný). Súbor je potom umiestnený v správe a odovzdaný Runtime Engine. Engine vytvorí novú inštanciu ClassLoader a načíta kód rozhrania (Java trieda alebo JAR) plus všetky súvisiace knižnice (JAR). ClassLoader má spôsob správania parent-last. To znamená, že každá trieda sa najprv vyhľadá v JAR archíve rozhrania alebo v súvisiacich archívoch. Ak trieda nebola nájdená,sú prehľadávané triedy runtime engine. Toto správanie umožňuje izolovať beh rozhrania z runtime engine. Keďže súvisiace archívy majú prednosť, rozhranímmôže byť pri behu využívaná knižnica, ktorá je už prítomná v runtime engine v rôznych verziách.

Akonáhle je ClassLoader inštanciovaný, inštancia triedy rozhranie je načítaná a spustená.

Spúšťacia správa (voliteľne obohatená o vstupný súbor) je odovzdaná ako parameter pre spustenie rozhrania.
Výstup behu rozhrania je zoznam odchádzajúcich správ. Každá správa je spojená s udalosťou. S udalosťou môže byť spojených viac lokalít. To znamená, že jedna správa môže byť dodaná do mnohých destinácií (napríklad FTP, JMS a e-mail). Za spracovanie odchádzajúcich správ je zodpovedný komponent Outbound. Každá správa je zhmotnená do súborov a žetónov, jeden pár súbor-token za každý cieľ. Žetóny sú jednoduché textové súbory popisujúce akciu doručenia.
Delivery Engine je samostatný proces, ktorý beží nezávisle na runtime engine. Pravidelne kontroluje existenciu nových odchádzajúcich správ. Každá odchádzajúca správa je potom v závislosti na konfigurácii odoslaná na cieľové miesto určenia. Cieľom môže byť FTP, JMS, e-mail, alebo HTTP.



Obrázok 4 - Runtime Engine

3.4 Dashboard

Dashboard je jediným miestom pre všetky potreby užívateľov. Poskytuje všetky informácie o rozhraniach a ich behoch, umožňuje všetky nastavenia konfigurácie a poskytuje bohatú sadu administrátorských nástrojov pre monitorovanie a správu celého prostredia.

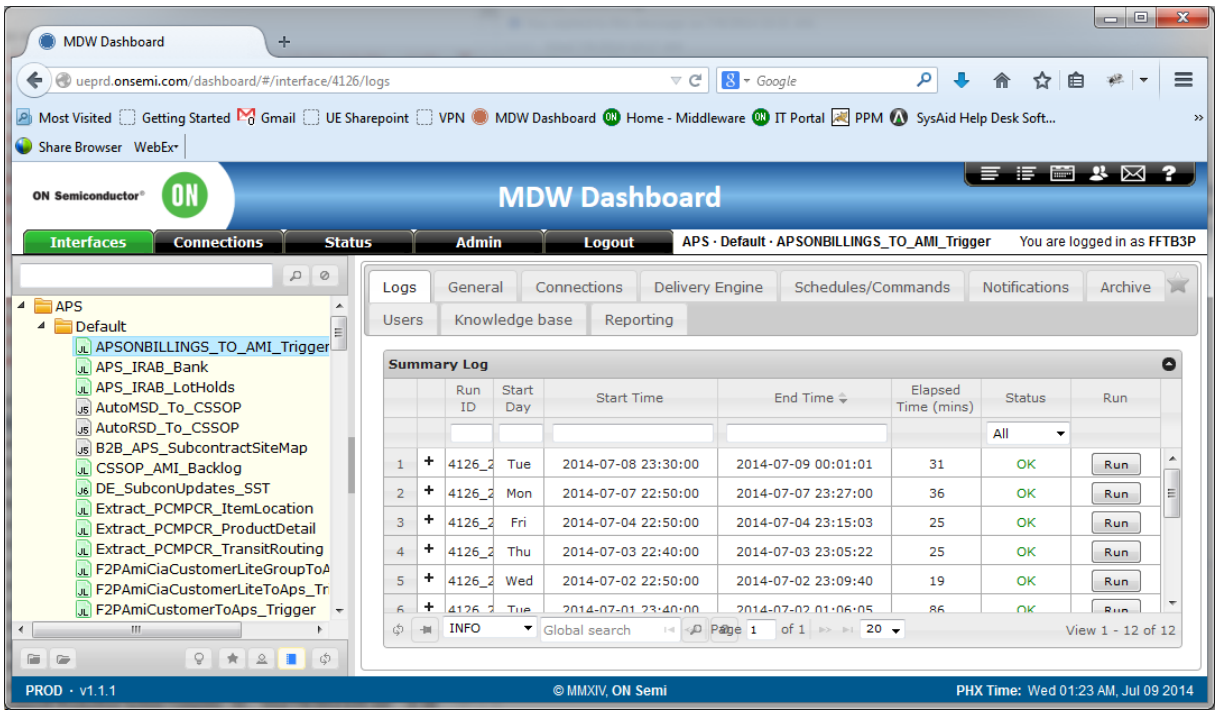
3.4.1 Karta Rozhrania

Karta Rozhrania je predvolená po prihlásení (pozri obrázok 5 – Rozhrania). Poskytuje komplexné informácie o všetkých rozhraniach. Na ľavej strane obsahuje menu so zoznamom všetkých rozhraní zoskupených podľa aplikačných skupín. Nad ponukou je vyhľadávacie pole. Umožňuje hľadať rozhranie podľa mena alebo podľa ID rozhrania. Pod menu je sada prepínačov pre rôzne možnosti zobrazenia (všetky rozhrania, vlastné rozhrania, obľúbené rozhrania, skryté rozhrania).
V pravej časti okna obsahuje informácie pre vybrané rozhranie.
Ďalej je v rozdelená podľa záložky:

- Záznamy - prvá záložka ukazuje záznamy o behu zoskupené podľa behov rozhrania. Záznamy možno triediť, filtrovať a prehľadávať.
- Hlavný panel - zobrazuje všeobecné informácie o rozhraní. Informácie je možné upraviť. Jednou z vlastností je nasadenie kódu rozhrania a súvisiacich archívov buď priamym nasadením súboru JAR alebo zavádzaním z prostredia Mercurial.
- Pripojenia - záložka zobrazuje a riadi všetky spojenia, ktoré sú k dispozícii pre rozhranie. V súčasnej dobe sú podporované databázy FTP, SFTP, JMS, HTTP a E-mail. Každému pripojeniu je priradený jedinečný názov v rámci rozhrania. Identifikátor rozhrania potom získa pripojenie pomocou týchto mien. Tento spôsob pri prenose rozhrania cez prostredie (DEV, INT, UAT a PROD) nemá vplyv na kód rozhraní. Jedinou zmenou je konfigurácia v Dashboard.
- Doručovanie- Nastavenie pre Delivery Engine. Vymedzenie udalostí pre dané rozhranie. Mapovanie činností na akcie dodania. Jedna udalosť môže byť mapovaná na rozmanité doručovacie akcie, napríklad FTP a E-mail. Rovnaká správa (obsah) je dodávaná na oba body.
- Plány/príkazy - Nastavenie súčasti Scheduler. Každej úlohe je potrebné najprv definovať príkaz - určený spúšťacou správou a typom potrebnej akcie. K dispozícii sú tri typy príkazov:
    - Aktívne - spustí rozhranie bez akýchkoľvek podmienok.
    - Databáza - pripája k určenej databáze a prevádzkuje daný SQL príkaz select. Ak príkaz SELECT vráti všetky riadky, rozhranie sa spustí.
    - Súbor - pripojenie k určenému FTP serveru a kontrola existencie daného súboru. Ak súbor existuje, spustí sa rozhranie.

  Plány definujú časové intervaly, kedy by sa práca mala začať. Plánovač používa Quartz cron syntax. Každý plán je definovaný cron výrazom a príkazom, ktorý má byť vykonaný.

- Upozornenia - definuje e-mailové akcie pre logovanie udalostí (TRACE, DEBUG, INFO, WARNING, ERROR a vlastné). Definícia špecifikuje cieľového príjemcu, predmet a telo. Predmet a telo môže využiť preddefinované premenné (napríklad názov rozhrania, beh ID, detail, zhrnutie).
- Archív - možnosť prechádzať archív prichádzajúcich a odchádzajúcich súborov. Archív je zoskupený podľa dátumu.
- Užívatelia - definícia bezpečnostných - užívateľských prístupových práv. Aplikácia umožňuje rozdielne úrovne kontroly. Prístupové práva sú nasledujúce: vlastné, beh, prehľadávať záznamy, zobraziť/modifikovať plány, zobraziť/upraviť oznámenia, zobraziť/zmeniť pripojenia, prezerať archív, nahrávať súbory, prezerať/upravovať informácie. Prístupové práva môžu byť stanovené na úrovni skupiny aplikácie alebo na úrovni rozhrania. Prístupové práva na aplikačnej úrovni sú dedičné a prepísané prístupovými právami rozhrania.
- Informácie o rozhraní – záložka by sa postupne mala stať hlavným zdrojom informácií o všetkých rozhraniach. Informácie možno definovať pre jedno rozhranie, alebo pre skupiny rozhraní.
- Hlásenia – správy o rozhraní na základe vopred definovaných šablón.

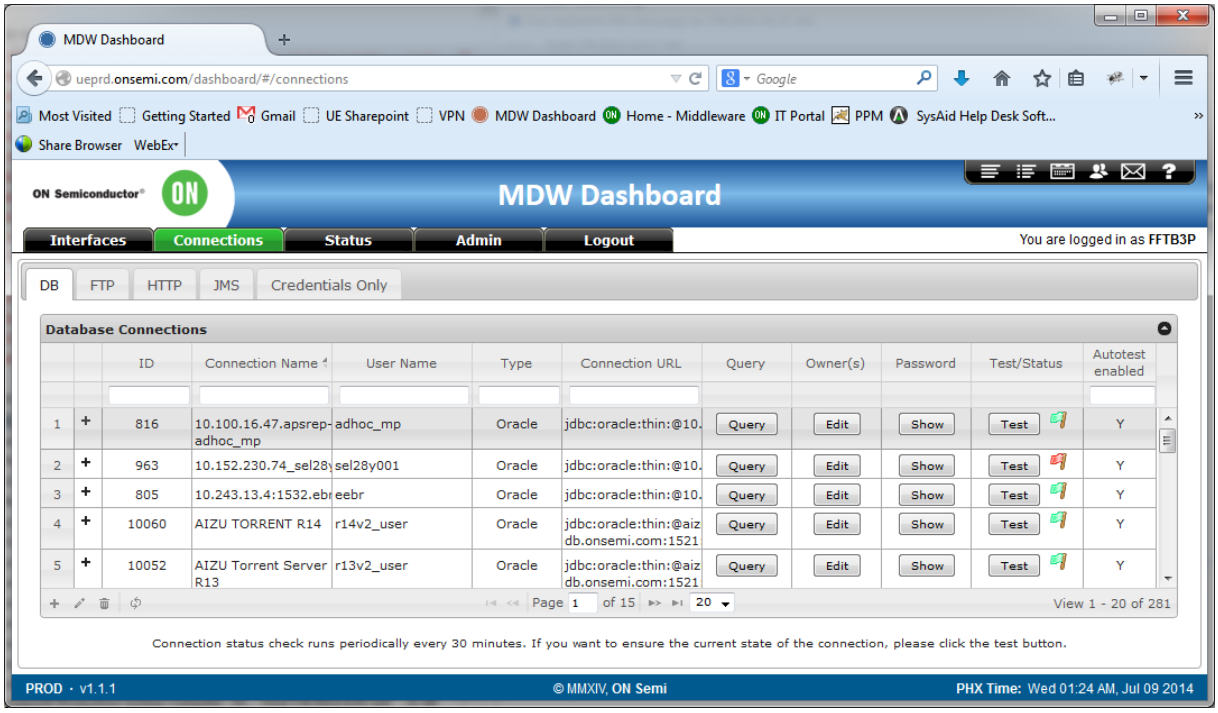


Obrázok 5 – Rozhrania

3.4.2 Karta Pripojenia

Táto záložka obsahuje definíciu globálnych dátových zdrojov (pozri obrázok 6 – Záložka pripojenia). Tieto zdroje dát môžu byť použité v definíciách rozhraní. Zdroje dát sú zoskupené podľa typu zdroja dát. V súčasnej dobe sú podporované nasledujúce typy zdrojov:

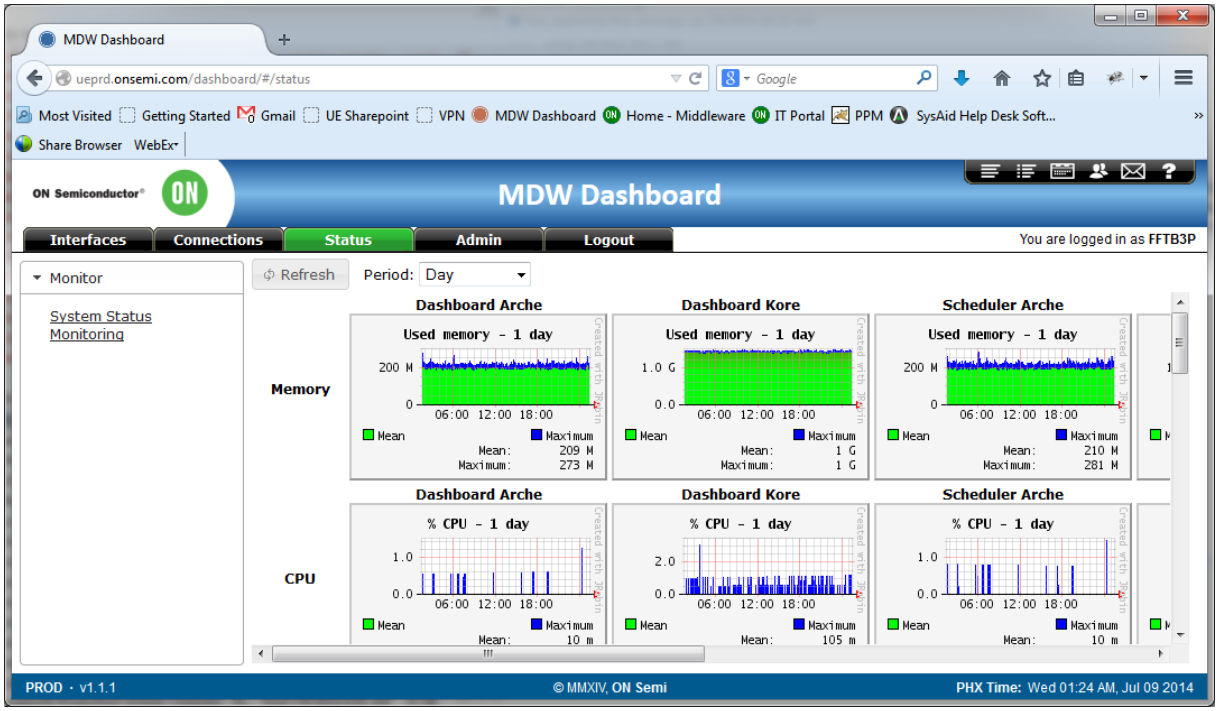
- DB - databáza, v súčasnej dobe sú podporované Oracle, MySQL, DB2, a SQL Server. Okrem spoločnej funkcie (triediť, filtrovať, upravovať) v zozname pripojenie umožňuje ďalšie funkcie: vyskúšať pripojenie, majitelia, zobraziť heslo (to je iba pre správcu a vlastníkov rozhrania), vyhľadávať v databáze s príkazom select.
- FTP - definícia dátových zdrojov FTP. Môže byť testovaná funkčnosť zdrojov dát, prehliadané heslo, a ukázaní majitelia.
- HTTP - zdrojov dát HTTP Post. Podporovaná funkčnosť - zobraziť majiteľov, heslo, test stavu.
- JMS - JMS témy a fronty. Podporované funkcie - ukázať majiteľov, heslo, test stavu.
- Iba osobné údaje - pripojenie len s užívateľským menom a heslom. Používa sa v FTP príkazy komponenty Scheduler. FTP server nie je súčasťou definície a musí byť určený pri použití pripojenia.



Obrázok 6 – Záložka pripojenia

3.4.3 Karta Status

Táto karta slúži na informovanie a zabezpečenie pohodlia užívateľa. Poskytuje prehľad o stave systému (pozri obrázok 7 - Status Tab). Odkaz „SystemStatus“ poskytuje prehľad o zdraví prostredia v členení podľa jednotlivých komponentov. Odkaz „Monitoring“ graficky zobrazuje všetky komponenty z rôznych hľadísk, napríklad CPU, pamäť, JDBC pripojenie, HTTP hity, SQL hity a iné.



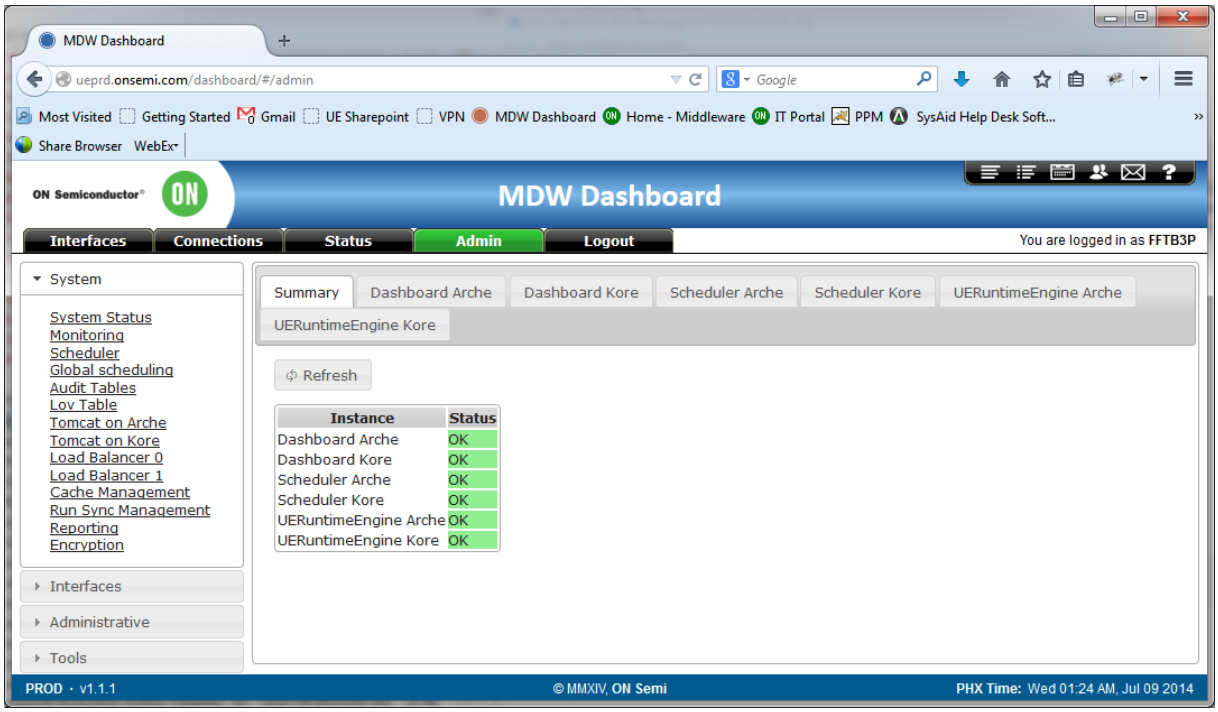
Obrázok 7 - Status Tab

3.4.4 Panel správcu

Správa systému, a teda k dispozícii je len pre správcov. Poskytované funkcie sú nasledujúce:

- Stav systému – podávanie prehľadu o stave zdravia prostredia členené podľa jednotlivých zložiek. Karty poskytujú viac informácií o jednotlivých komponentoch a farebne ukázujú zdravie (zelená = OK, červená = zlé).
- Monitoring – JavaMelody (opensource knižnica), monitorovanie celého prostredia. Pomerne komplexné informácie s históriou.
- Globálne plánovanie – informácie o plánovači (stav, čas štartu, počet realizovaných behov) plus ovládanie spustenia, zastavenia a znovuspustenia.
- Audit tabuľky – prístup k histórii všetkých zmien konfigurácie v prostredí.
- Lov tabuľka – parametre prostredia používané v konfigurácii. Používa sa vo všetkých aspektoch prostredia.
- Tomcat – prístup ku konzole Probe Tomcat serveru.
- Load Balancer – prístup k Apache Mod_jk konzole vyrovnávača zaťaženia.

- Cache Management – pohľad na zoznam vyrovnávacích pamätí so schopnosťou znovu načítať dáta pre všetky alebo jednotlivé pamäte.
- Run Sync Management – prehliadanie sychronizačných objektov. Tieto objekty sú používané pre synchronizáciu napr. runtime (ktorý z nich by mal spustiť rozhranie) a pre singleton implementácie (naraz nie je možné spustiť viac inštancií rozhrania). Tieto objekty môžu byť v prípade chyby odstránené.
- Reporting – spravovanie šablóny hlásení, ktoré by mohli byť použité pre jednotlivé rozhrania.
- Šifrovanie – riadenie súkromných a verejných kľúčov PGP, používané v rozhraniach.
- Novinky/Oznamy – spravovanie zoznamu oznámení, ktoré môžu byť zobrazené cez ikonu Novinky v záhlaví. Novinky môžu byť alternatívne zobrazené v päte stránky ako plávajúci pruh.
- Poznámky k vydaniu – spravovanie poznámky k vydaniu. Môžu byť zobrazené pomocou ikony poznámkach k vydaniu na hlavičke.
- Relogin – prihlásenie sa pod iným užívateľským účtom. Užitočné pri riešení problémov so zákazníkom, je možné vidieť presne to, čo vidí zákazník a navigovať ho.
- Vytvoriť užívateľa – vytvoriť nového užívateľa v systéme s východiskovým nastavením.
- Správca užívateľov – spravovať zoznam administrátorov v systéme.



Obrázok 8 - Admin Tab

3.5 Scheduler

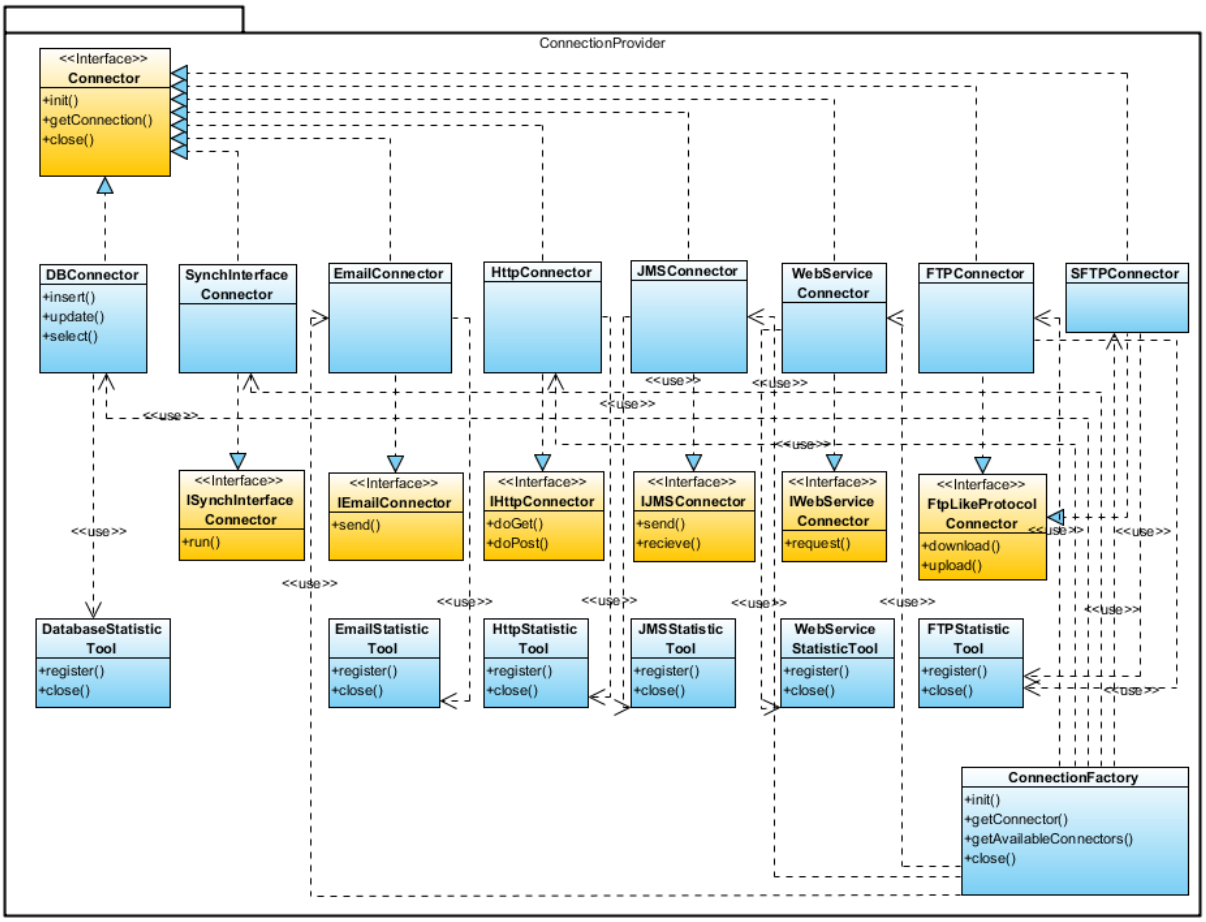
Komponent Scheduler je aplikácia spustená v kontajneri J2EE aplikácií (Tomcat). Je založená na Quartz Route - open source Java knižnici pre plánovanie behov. Aplikácia využíva middleware databázu ako úložisko konfigurácie.

Hlavnou požiadavkou pre framework je spoľahlivosť. Pre vysokú dostupnosť middleware prostredia využíva dve inštancie plánovača. Oba plánovače spúšťajú všetky úlohy a až vykonávajúci Runtime Engine zaisťuje, že každá úloha sa spustí iba raz.

Kapitola 4
Implementácia

4.1 Požiadavky
Vývoj knižnice na prístup k dátovým zdrojom výrazne podliehal potrebám a požiadavkám spoločnosti ON Semiconductor, pre ktorú bola knižnica vyvíjaná. Základnou požiadavkou je vyjsť z už používanej knižnice poskytujúcej prístup k databázam a obmedzenú funkcionalitu na prístup k FTP serverom. Súčasnú funkcionalitu knižnice je potrebné rozšíriť a doimplementovať ďalšie komponenty umožňujúce využitie ďalších dátových zdrojov. Ďalšou požiadavkou je doplniť štatistický modul, ktorý ukladá dôležité údaje o používaní jednotlivých častí aplikácie do databázy.
Pri návrhu riešenia je potrebné dbať na jednoduchú rozšíriteľnosť knižnice a čo najľahšie používanie. Aplikácia musí podporovať protokoly SQL, FTP a SFTP, SMTP, HTTP, SOAP, JMS a umožňovať spúšťanie iných rozhraní v rámci systému.

4.2 Objektový návrh
Výsledkom tohto návrhu je priložený obrázok 9 – Diagram tried. Pri práci s aplikáciou je potrebné, aby bolo možné konektory k jednotlivým protokolom získať z rozhrania v čo najmenšom počte krokov. Preto je zavedená trieda Connection Factory, ktorá by nastavenie parametrov konektorov zvládala automaticky a používateľ môže iba požiadať o inštanciu konkrétneho konektoru, s ktorým môže rovno pracovať. Jednotlivé konektory sa od seba výrazne líšia, a tak je pre každý jeden zavedené samostatné rozhranie, ktoré popisuje jeho funkcie. Všetky však musia mať metódy pre inicializáciu, uzatvorenie a získanie základných nastavení, preto je zavedené druhé rozhranie, jednotné pre každý konektor. Štatistické moduly sú vyvedené do samostatných tried pre oddelenie rozdielnej funkcionality a prípadnú budúcu ľahšiu extrakciu.



Obrázok 9 – Diagram tried

4.3 Komponenty

4.3.1 Connection Factory

Connection Factory je hlavnou triedou knižnice a slúži ako iniciálna trieda pre vytváranie jednotlivých konektorov. Pred použitím triedy je ju potrebné inicializovať pomocou metódy *init()*, ktorá pre správny beh vyžaduje inštanciu triedy JdbcTemplate z Java balíku Spring. Ako konštruktor konektorov funguje metóda *getConnector()*, ktorá ako parametre vyžaduje triedu požadovaného konektoru, triedu nesúcu informácie o pripojení daného typu priradenú rozhraniu, ktoré knižnicu používa a voliteľne inštanciu triedy LogDispatcher. Táto je potrebná iba v prípade, ak chceme zabezpečiť spätnú propagáciu záznamov do aplikácie Dashboard. Zoznam dostupných konektorov je možné získať zavolaním metódy getAvailableConnectors(), ktorá vráti zoznam názvov podporovaných konektorov. Po skončení práce je potrebné konektory ukončiť pomocou metódy releaseAll().

4.3.2 DB konektor

SQL je jazyk špeciálne navrhnutý pre prácu s databázami. Bol predstavený v roku 1970. Prvá implementácia tohto jazyka sa objavila v roku 1979 a v súčasnosti je priemyselným štandardom. Formálne je jeho podoba riadená štandardizačnou organizáciou ISO. Umožňuje vytvárať databázy, pridávať do nich dáta, udržiavať ich

a získavať vybrané časti dát. SQL umožňuje pracovať s dátami na logickej úrovni. Spracúvanie riadkov netreba riešiť jeden po druhom ani starať sa o to, ako sú fyzicky uložené alebo vyvolané. SQL poskytuje príkazy pre rad úloh vrátane získavania dát, vkladania, aktualizácie a mazania riadkov v tabuľke, vytvorenia, nahrádzania, pozmeňovania a mazania objektov, riadenia prístupu k databáze a jej objektom. Záruka konzistencie databázy a integrity SQL zjednocuje všetky predchádzajúce úlohy do jedného konzistentného jazyka **[18].**
Trieda DBConnector obsahuje implementáciu DB konektoru. Je zároveň jedinou triedou konektoru, ktorá bola naimplementovaná a poskytnutá v pôvodnej knižnici ConnectorProvider.
Využíva framework Spring JDBC, ktorý sa stará o všetky detaily, počnúc otvorením spojenia, cez pripravenie a spustenie príkazu SQL, spracovanie výnimiek, spracovanie transakcií až po ukončenie pripojenia. Ostáva len definovať parametre pripojenia, zadať príkaz SQL, ktorý má byť vykonaný a spracovať dáta načítané z databázy **[19].** Konektor využíva JdbcTemplate, triedu, ktorá spravuje všetku komunikáciu databáz a spracovanie výnimiek.
Trieda DBConnector implementuje rozhranie Connector. Toto rozhranie predpisuje jednotnú formu pre všetky konektory knižnice. Obsahuje metódy pre inicializáciu konektoru, nastavenie pripojenia, čo je objekt, v rámci ktorého sú uložené údaje potrebné na pripojenie ku konkrétnemu dátovému zdroju a prednastavená východisková správa. Ďalej zahŕňa metódy pre nastavenie ďalších objektov a identifikátorov požadovaných systémom a metódu pre ukončenie práce s konektorom.
Trieda ďalej poskytuje funkcie pre vykonávanie SQL príkazov, volanie funkcií a procedúr, dávkové vkladanie riadkov, výber listov, reťazcov, číselných hodnôt a dátumov, výber so spracovaním externým objektom a automatické alebo manuálne potvrdzovanie transakcií.
Trieda DatabaseStatisticTool slúži na zbieranie štatistík o použití DB konektoru a ich následný zápis do databázy.
Na zaznamenávanie dát z jednotlivých prenosov slúži metóda register(), ktorá ako parametre vyžaduje názov, pod ktorým je v systéme uložený set SQL príkazov, typ operácie a počet riadkov, ktorými je manipulované. Podporované sú operácie vkladania značené reťazcom „insert“, úpravy označené reťazcom „update“ a operácia výberu, ktorá je značená reťazcom „select“. Štatistiky sú do databázy ukladané po ukončení práce s konektorom, kedy je na štatistickej triede zavolaná metóda close().

4.3.3 FTP konektor

FTP je skratka pre File Transfer Protocol. Ide o protokol používaný na prenos súborov medzi FTP hostiteľským serverom a klientským počítačom prostredníctvom internetu. Najčastejšie je využívaný na sťahovanie súborov z World Wide Webu. Ide o alternatívnu voľbu pre sťahovanie a upload dát k protokolu HTTP. Pre zahájenie prenosu je potrebné poznať adresu servera, užívateľské meno, heslo a číslo portu.

Anonymní používatelia môžu súbory iba sťahovať, pre nahrávanie je potrebné prihlásenie. Pripojenie nikdy nezostáva otvorené, akonáhle je prenos dát dokončený, pripojenie je ukončené. Pre každú ďalšiu interakciu je spustený nový prenos **[20].**
Protokol FTP podporuje dva režimy pripojenia - ASCII a binárny. Režim je určený v počiatočnej fáze všetkých transakcií FTP serverom. FTP prenáša súbory medzi systémami vždy s použitím jedného z týchto dvoch režimov. Režim ASCII sa používa výhradne k prenosu textu a HTML. Binárny režim zabezpečuje prenos súborov v binárnej forme. Binárne súbory nemožno odoslať prostredníctvom ASCII režimu, inak dôjde k problémom v konzistencii **[20].**
Trieda FtpConnector obsahuje implementáciu FTP konektoru, na čo je využitý framework Apache Commons FTPClient. Tento zapúzdruje všetky funkcie potrebné na ukladanie a načítanie súborov z FTP serveru, stará sa o všetky detaily nižšej úrovne interakcie s FTP serverom a poskytuje pohodlné rozhranie vyššej úrovne **[21].**
Pred začatím prenosu za pomoci knižnice FTPClient je potrebné zavolať metódu connect() a po dokončení metódu disconnect(). Následne je potrebné skontrolovať FTP kód odpovede, či pripojenie bolo úspešné. Konvenciou pre všetky FTP metódy príkazov vo frameworku je buď vrátiť logickú alebo nejakú inú hodnotu. Pre metódy vracajúce logickú hodnotu je vrátená hodnota true značiaca úspešné dokončenie prenosu a hodnota false pre prenosy, ktoré končia chybovým stavom alebo poruchou. Metódy vracajúce inú ako boolean hodnotu vrátia odpoveď obsahujúcu údaje vyššej úrovne alebo hodnotu null, ak príkaz viedol k chybovému stavu alebo poruche. Okrem toho sú z bezpečnostných dôvodov všetky dátové pripojenia ku klientovi overené, aby sa zabezpečilo, že pochádzajú zo správneho zdroja. Ak je dátové pripojenie nadviazané neočakávané stranou, je vyhodená výnimka. Toto správanie je však možné zmeniť.
Trieda implementuje rozhrania Connector a FtpLikeProtocolConnector. Rozhranie Connector predpisuje jednotnú formu pre všetky konektory knižnice a obsahuje metódy pre inicializáciu konektoru, nastavenie pripojenia, čo je objekt, v rámci ktorého sú uložené údaje potrebné na pripojenie ku konkrétnemu dátovému zdroju a prednastavené východiskové súbory, metódy pre nastavenie ďalších objektov a identifikátorov požadovaných systémom a metódu pre ukončenie práce s konektorom.Konkrétne parametre pripojenia môžu byť používateľom jednoducho nastavené z webového rozhrania prostredia – Dashboard.
Implementácia rozhrania FtpLikeProtocolConnector vyžaduje, aby trieda obsahovala metódu pre získavanie vstupných súborov z prostredia Runtime Engine, metódu pre vytváranie a prenos výstupných správ, metódy pre nahrávanie jedného alebo viacerých súborov do vzdialenej lokality, vytvorenie súboru z textového reťazca a jeho následné odoslanie, determináciu, či daný súbor v lokalite existuje, metódu pre premenovávanie súborov, metódu pre sťahovanie súborov z danej lokality a metódu na odstránenie dočasných súborov.
Nad rámec rozhraní dokáže trieda FtpConnector rozpoznať regulárny výraz ako názov súboru a na jeho základe pracovať so súbormi, ktorých názvy patria do množiny určenej týmto výrazom. Z dôvodu ich použitia vo firemnom prostredí a ich

obmedzeniam bolo ďalej nutné doimplementovať rozpoznávanie IBM Mainframe a HP NonStop (Tandem) systémov.
Pri použití konektoru je potrebné mať na pamäti, že FTP server sa môže rozhodnúť predčasne ukončiť pripojenie, ak je klient nečinný po dobu dlhšiu ako určité časové obdobie (obvykle 900 sekúnd).
Trieda FtpStatisticTool slúži na zbieranie štatistík o prenosoch realizovaných za pomoci FTP konektoru a ich následný záznam do databázy. Pre zber jednotlivých dát slúži metóda register(), ktorá ako parametre dostáva názov súboru, jeho veľkosť v bajtoch a textový reťazec operácia značiaci smer prenosu dát. V súčasnosti sú podporované operácie s obsahom reťazca „read" pre dáta, ktoré sú systémom prijímané a reťazcom „write" pre odosielané dáta. Po zavolaní metódy sú štatistiky ihneď odosielané do databázy, vzhľadom k tomu, že je žiadané mať zachované údaje o každom pohybe súborov v rámci systému a relatívnej časovej nenáročnosti prístup k databáze v porovnaní s dobou prenosu súboru.

4.3.4 SFTP konektor

Protokol SFTP, SSH File Transfer Protocol, predstavuje bezpečnú alternatívu k FTP a je protokolom, ktorý slúži na vytvorenie bezpečného spojenia medzi vzdialeným serverom a počítačom.Secure Shell využíva šifrovanie pomocou verejného kľúča a poskytuje jednoznačné overovanie identity používateľov a bezpečnú šifrovanú komunikáciu cez internet. Riadiace i dátové pripojenia sú zašifrované medzi klientom a serverom FTP, čo umožňuje prenášať heslá a iné citlivé informácie bezpečne po sieti. Pripojenie na SFTP servery je ustanovené buď overením hesla, alebo pomocou verejných a súkromných kľúčov **[22].**
Pre prihlásenie za pomoci hesla používateľ potrebuje iba používateľské meno a heslo pre prístup na SFTP server. Ďalšou metódou je použitie páru kľúčov. Dvojice verejných a súkromných kľúčov sú generované a verejné kľúče sú uložené na SFTP serveri. Klient sa prihlási so súkromným kľúčom, ktorý sa na serveri počas prihlasovania overí a pokiaľ kľúče zodpovedajú, klient SFTP získa prístup do systému. Privátny kľúčmôže byť chránený heslom. SFTP formátuje príkazy a dáta do špeciálnych paketov a odosiela ich prostredníctvom jediného pripojenia. Nepoužíva samostatné kanály ako FTP. To eliminuje potrebu otvorenia širokého rozsahu portov, ako je to bežné pre FTP servery **[22].**
Trieda SFTPConnector obsahuje implementáciu SFTP konektoru, pričom je využitý framework Apache Commons VFS. Pre prenos súborov je používaný objekt StandardFileSystemManager, ktorý zabezpečuje celú réžiu prenosu **[23].**
Rovnako ako pri FTP konektore trieda implementuje rozhrania Connector a FtpLikeProtocolConnector.
Rozhranie Connector obsahuje metódy pre inicializáciu konektoru, nastavenie pripojenia, metódy pre nastavenie ďalších objektov a identifikátorov požadovaných systémom a metódu pre ukončenie práce s konektorom. Konkrétne parametre pripojenia môžu byť používateľom opäť nastavené z webového rozhrania prostredia – Dashboard.

Implementácia rozhrania FtpLikeProtocolConnector vyžaduje, aby trieda obsahovala metódu pre získavanie vstupných súborov z prostredia Runtime Engine, metódu pre vytváranie a prenos výstupných správ, metódy pre nahrávanie jedného alebo viacerých súborov do vzdialenej lokality, vytvorenie súboru z textového reťazca a jeho následné odoslanie, determináciu, či daný súbor v lokalite existuje, metódu pre premenovávanie súborov, metódu pre sťahovanie súborov z danej lokality a metódu na odstránenie dočasných súborov. Pri pripájaní na server je možné prihlásiť sa za pomoci kombinácie mena a hesla alebo pomocou RSA kľúča. Nad rámec rozhraní dokáže trieda SFTPConnector rozpoznať regulárny výraz ako názov súboru a na jeho základe pracovať so súbormi, ktorých názvy patria do množiny určenej týmto výrazom a vytvárať priečinky na vzdialenom serveri.
Pre zhromažďovanie štatistík je využitá trieda FtpStatisticTool, ktorá je popísaná v kapitole 4.3.3. Toto riešenie môže byť využité vzhľadom k tomu, že SFTP a FTP konektor sú si značne podobné a požadované údaje o prenášaných súboroch sa nijak nelíšia.

4.3.5 SMTP konektor

Elektronická pošta je jednou z najpopulárnejších služieb siete v dnešnej dobe. Väčšina e-mailových systémov, ktoré odosielajú poštu, používajú jednoduchý protokol pre prenos pošty (SMTP) na odosielanie správ z jedného servera na iný. Správy môžu byť potom príjmané pomocou e-mailového klienta buď prostredníctvom protokolu Post Office Protocol (POP) alebo protokolu Internet Message Access Protocol (IMAP). POP definuje podporu jednej schránky pre každého používateľa. IMAP poskytuje podporu pre viacero schránok pre každého používateľa a navyše poštová schránka môže byť zdieľaná viacerými užívateľmi.
SMTP sa používa ako spoločný mechanizmus pre prepravu elektronickej pošty nad rodinou protokolov TCP/IP v aplikačnej vrstve **[24].**
SMTP klient vytvorí TCP spojenie k SMTP procesu servera na vzdialenom počítači a pokúsi sa odoslať poštu. Server čaká na pripojenie na určitý port (obvykle 25), na ktorý sa klient pripojí. Ak je TCP spojenie úspešné, dva procesy spustia jednoduchý dialóg požiadavka - odpoveď, definovaný pomocou protokolu SMTP, v ktorom klient proces prenesie mailové adresy pôvodcu a príjemcov správy.
Keď server spracuje prijaté adresy, klient začne s prenosom obsahu e-mailovej správy. Správa musí obsahovať hlavičku správy a text správy, ktoré sú formátované v súlade s štandardom RFC 822. MIME protokol, skratka pre Multipurpose Internet Mail Extensions, definuje spôsob, akým sú súbory pripojené k správam SMTP. Správa prenesená cez SMTP je odovzdaná vzdialenému serveru alebo je doručená do poštovej schránky na lokálnom serveri. Protokoly POP3 alebo IMAP potom umožňujú používateľom sťahovať poštu, ktorá je na lokálnom serveri uložená **[24].**
Trieda EmailConnector obsahuje kompletnú realizáciu SMTP konektoru, pričom je využitý balík JavaMail API, ktorý je štandardnou súčasťou Java Platform, Enterprise Edition.
Je to sada abstraktných rozhraní, ktorá modeluje poštový systém a poskytuje funkcie

pre čítanie, písanie a odosielanie elektronických správ.
Obsahuje rozhrania, ktoré sa používajú na vytvorenie rozhrania pre systém zasielania správ a niekoľko tried, ktoré implementujú RFC822 a MIME štandardy. MIME, definuje obsah toho, čo sa prenáša: formát správ, príloh, a podobne **[25].**
Trieda implementuje rozhrania Connector a IEmailConnector. Rozhranie Connector predpisuje jednotnú formu pre všetky konektory knižnice a obsahuje metódy pre inicializáciu konektoru, nastavenie pripojenia, čo je objekt, v rámci ktorého sú uložené údaje potrebné na pripojenie ku konkrétnemu dátovému zdroju a prednastavená východisková správa, metódy pre nastavenie ďalších objektov a identifikátorov požadovaných systémom a metódu pre ukončenie práce s konektorom. Konkrétne parametre pripojenia môžu byť používateľom jednoducho nastavené z webového rozhrania prostredia – Dashboard.
Implementácia rozhrania IEmailConnector vyžaduje, aby konektor obsahoval metódy pre zaslanie správy tak, ako bola nakonfigurovaná v objekte pripojenia, metódy pre nastavenie odosielateľa správy, pridanie alebo odobratie príjemcu správy, príjemcu správy v kópii alebo príjemcu správy v skrytej kópii a metódy pre nastavenie hlavičiek, predmetu a obsahu správy vrátane pridávania príloh. Nad rámec rozhraní konektor pridáva metódu na nastavenie podpisu, čo je textový reťazec, ktorý sa automaticky pridá na koniec každej správy a metódy pre získanie jednoduchých alebo podrobných štatistík o zaslaných správach.
Trieda EmailStatisticTool zhromažďuje štatistické údaje o jednotlivých správach prenesených pomocou SMTP konektoru. Zbieranie týchto dát zabezpečuje metóda register(), ktorej je ako parameter daný reťazec s adresami odosielateľov, reťazec s adresami prijímateľov, reťazec s adresami prijímateľov v kópii, predmet správy a prvých 1000 znakov obsahu správy, čo je pre účely štatistík dostačujúce. Prípadné prílohy nie sú zaznamenávané. Po zavolaní metódy sú dáta uložené do databázy, čo vzhľadom k typicky nízkemu objemu zasielaných správ nijako výrazne nepredlžuje dobu behu rozhrania, ktoré SMTP konektor využíva.

4.3.6 HTTP konektor

Hypertext Transfer Protocol (HTTP) je protokol aplikačnej úrovne pre distribuované a spolupracujúce informačné systémy, ktorý umožňuje základný prístup k dátam získaným z rôznych aplikácií. Protokol HTTP je protokol pracujúci na princípe požiadavky a následnej odpovede.Klient na server odošle požiadavku, ktorá obsahuje typ metódy požiadavky, URI, čo je reťazec znakov používaný na identifikáciu názvu zdroja, verziu protokolu a následne správu podobnú štandardu MIME, obsahujúcu modifikátory požiadavky, informácie o používateľovi a prípadné ďalšie dáta. Server odpovie stavovým riadkom, ktorý obsahuje verziu protokolu pôvodnej správy a kód značiaci úspešnú alebo chybnú operáciu. Tento riadok je nasledovaný správou podobnou štandardu MIME, ktorá obsahuje informácie o serveri, obsah požadovaného objektu a metainformácie.
HTTP komunikácia sa zvyčajne uskutočňuje cez TCP/IP spojenia. Predvolený port je

TCP 80, ale môžu byť použité i iné porty. To však nevylučuje použitie HTTP nad iným protokolom na internete alebo v iných sieťach **[26].** Predpokladom pre použitie je iba spoľahlivosť dopravy. Môže byť použitý akýkoľvek protokol, ktorý poskytuje záruku takéhoto spojenia.
Trieda HttpConnector obsahuje kompletnú realizáciu HTTP konektoru.
Pre realizáciu komponentu HTTP konektor je využitý framework Apache Commons HttpComponents. Je to set komponentov, ktoré sú použité na vykonávanie základných operácií za využitia protokolu HTTP, ale dostačuje k rozvoju plne funkčnej klientskej, prípadne serverovej aplikácie. Zároveň má jednotné aplikačné rozhranie pre budovanie ako synchrónnych, tak aj asynchrónnych HTTP služieb. Využitá je najmä knižnica HttpClient, ktorá zabezpečuje podporu klientských funkcií celého frameworku **[27].**
Trieda implementuje rozhrania Connector a IHttpConnector. Rozhranie Connector podobne ako pri predchádzajúcich konektoroch obsahuje metódy pre inicializáciu konektoru, nastavenie pripojenia, metódy pre nastavenie ďalších objektov a identifikátorov požadovaných systémom a metódu pre ukončenie práce s konektorom. Konkrétne parametre pripojenia môžu byť používateľom jednoducho nastavené z webového rozhrania prostredia – Dashboard.
Implementácia rozhrania IHttpConnector požaduje, aby konektor obsahoval metódy pre odosielanie požiadaviek metód GET a POST, pričom niektorým premenným v požiadavke typu ${premenná} môžu byť priradené hodnoty buď z objektu alebo mapy. Pri mapovaní z objektu sa musí názov premennej zhodovať s názvom atribútu objektu. Pri mapovaní z mapy sa musí názov premennej zhodovať s kľúčom. Odpoveď webového serveru môže byť vrátená ako textový reťazec alebo namapovaná do objektu. V tom prípade je potrebné metóde definovať triedu objektu, ktorý má byť vrátený, a atribúty tejto triedy sa musia zhodovať s názvami XML tagov v odpovedi. Ďalej toto rozhranie požaduje implementáciu metód zabezpečujúcich sťahovanie súborov, nahrávanie súborov a odosielanie výstupných správ. Nad rámec rozhraní trieda implementuje metódy na pridávanie a odstraňovanie hlavičiek požiadavky, výber úrovne dôvery k certifikátu zdroja a metódu pre autorizáciu.
Trieda HttpStatisticTool slúži na zaznamenávanie štatistík o prenose dát za pomoci HTTP konektoru, ktoré následne vkladá do databázy. Ukladá samostatné záznamy pre každú URL adresu, ku ktorej bolo pristúpené a priraďuje k nim objem stiahnutých a odoslaných dát. Pre zhromažďovanie záznamov slúži metóda register(), ktorá ako svoje parametre vyžaduje adresu URL, textový reťazec značiaci typ operácie a objem prenesených dát v bajtoch. V súčasnej dobe je povolená operácia sťahovanie, ktorá je označená reťazcom „download“ a operácia nahrávanie označená reťazcom „upload“. Pre akúkoľvek inú operáciu štatistický nástroj vypíše varovanie, že operácia je nepodporovaná a nebola zaznamenaná. Počas behu programu sú štatistiky uchovávané v pamäti, do databázy sa z dôvodu časovej optimalizácie zapíšu až po uzavretí štatistického nástroja zavolaním metódy close().

4.3.7 Konektor webových služieb

Webové služby sú softvérové komponenty, ktoré komunikujú pomocou protokolu HTTP a zasielaním správ založených na XML.

Webové služby sú navrhnuté tak, aby boli prístupné iným aplikáciám. Líšia sa v komplexnosti od jednoduchých operácií ako kontrola zostatku bankového účtu po zložité procesy, systémy riadenia vzťahov so zákazníkmi alebo ERP systémy.Webové služby sú založené na XML a troch ďalších kľúčových technológiách: WSDL, SOAP a UDDI **[28].** Pred vytvorením webovej služby vývojári definujú jej podobu v dokumente WSDL, ktorý opisuje polohu služby na webe a funkciách, ktoré služba poskytuje. Informácie o službe potom môžu byť zapísané v registri UDDI, ktorý umožňuje používateľom webových služieb hľadať a nájsť služby, ktoré potrebujú. Tento krok je voliteľný. Vhodný je v prípade, ak spoločnosť chce, aby jej webové služby mohli byť objavené externými používateľmi. Na základe informácií v registri UDDI vývojár využije inštrukcie v WSDL dokumente na zostavenie SOAP správy, ktorá slúži na samotnú výmenu dát zo služby prostredníctvom HTTP.

WSDL, SOAP, UDDI sú založené na XML. WSDL (Web Services Description Language) je formát založený na XML pre opis webových služieb. Klienti, ktorí chcú získať prístup k webovej službe, môžu čítať a interpretovať jeho WSDL súbor, aby sa dozvedeli o umiestnení služby a jej dostupných operáciách.

WSDL dokument opisuje webovú službu ako zmluvu medzi klientom a serverom webových služieb. Dodržiavaním tohto kontraktu sú si poskytovateľ a používateľ schopní vymieňať dáta štandardným spôsobom, bez ohľadu na platformu a aplikácie, s ktorými operujú. Pretože protokol HTTP je podporovaný všetkými webovými servermi a prehliadačmi, SOAP správy je možné posielať medzi aplikáciami bez ohľadu na ich platformu alebo programovací jazyk. Táto vlastnosť poskytuje webovým službám ich charakteristickú vlastnosť - interoperabilitu.

Pretože klient a server musia dodržiavať zmluvu WSDL pri vytváraní správ SOAP, správy sú zaručene kompatibilné.

Trieda WebServiceConnector obsahuje kompletnú implementáciu konektoru webových služieb, na čo je využitý balík javax.xml.soap. Tento poskytuje rozhranie API pre vytváranie a budovanie SOAP správ aumožňuje vytvárať pripojenie typu point-to-point k určitému koncovému bodu, tvoriť správy SOAP, pridávať obsah do hlavičky a tela správ SOAP, pridávať ku správam prílohy a odosielať správy.Ďalej je možné pristupovať k už vytvoreným SOAP správam, modifikovať ich a získavať ich obsah **[29].** Okrem poskytovania API, balík javax.xml.soap rozširuje niektoré triedy v balíčku org.w3c.dom. To znamená, že SOAP správy sú reprezentované ako DOM dokument a je teda možné ich manipulovať tými aplikáciami, nástrojmi a knižnicami, ktoré spracovávajú DOM **[29].**

Trieda implementuje rozhrania Connector a IWebserviceConnector. Rozhranie Connector, tak ako v predchádzajúcich konektoroch, predpisuje jednotnú formu pre všetky konektory knižnice a obsahuje metódy pre inicializáciu konektoru, nastavenie pripojenia, metódy pre nastavenie ďalších objektov a identifikátorov požadovaných

systémom a metódu pre ukončenie práce s konektorom. Konkrétne parametre pripojenia môžu byť používateľom znovu jednoducho nastavené z webového rozhrania prostredia – Dashboard.
Rozhranie IWebServiceConnector požaduje, aby konektor obsahoval metódy pre odosielanie požiadavky na webovú službu, pričom niektorým premenným v zdrojovej správe typu ${premenná} môžu byť priradené hodnoty buď z objektu, alebo mapy. Pri mapovaní z objektu sa musí názov premennej zhodovať s názvom atribútu objektu. Pri mapovaní z mapy sa musí názov premennej zhodovať s kľúčom. Odpoveď webovej služby môže byť vrátená ako textový reťazec, alebo namapovaná do objektu. V tom prípade je potrebné metóde definovať triedu objektu, ktorý má byť vrátený a atribúty tejto triedy sa musia zhodovať s názvami XML tagov v odpovedi. WebServiceStatisticTool je trieda zhromažďujúca štatistiky o používaní WebService konektoru. Dáta sú zaznamenávané volaním metódy register(), ktorá ako parametre berie názov koncového bodu pripojenia, prvých 1000 znakov požiadavky a prvých 1000 znakov odpovede webovej služby. Štatistiky sú pri každom volaní metódy zaznamenávané do databázy.

4.3.8 JMS konektor

JMS je set rozhraní a príslušnej sémantiky, ktorá definuje, ako klient pristupuje k možnostiam zasielania správ v rámci podniku. Aplikácia JMS sa skladá zo sady definovaných foriem správ a súboru klientov, ktorí si ich vymieňajú. Každý takýto systém poskytuje nejaký spôsob adresovania správ. Každý poskytuje spôsob, ako vytvoriť správu a naplniť ju dátami. Niektoré systémy sú schopné vysielať správy do mnohých destinácií, iné podporujú odoslanie správy len do jedného cieľového miesta. Niektoré systémy poskytujú asynchrónny príjem správ, teda správy sú doručované klientovi, ako sa objavia. Iné podporujú len synchrónny príjem – klient si musí každú správu vyžiadať **[30].**
Každý systém zasielania správ zvyčajne poskytuje rad ďalších nastavení, ktoré môžu byť zvolené pre jednotlivé správy, ako napríklad priorita doručenia správy, čas, dokedy je správa aktuálna alebo či je požadovaná odpoveď.
Keďže zasielanie správ je typu peer-to-peer, všetci užívatelia JMS sú označovaní ako klienti. Zasielanie správ značí asynchrónnu komunikáciu medzi podnikovými aplikáciami. Správy sú požiadavky alebo udalosti a sú spracovávané podnikovými aplikáciami. Obsahujú dôležité informácie potrebné pre koordináciu systémov. Telom správ sú presne formátované údaje, ktoré opisujú špecifické obchodné akcie. Prostredníctvom výmeny týchto správ každá aplikácia sleduje priebeh činností podniku. Všeobecne platí, že každý klient môže komunikovať s akýmkoľvek iným klientom v rámci systému.Každý klient sa pripája k agentovi na zasielanie správ, ktorý poskytuje niekoľko možností vytvárania, odosielania a prijímania správ **[30].**
Trieda JMSConnector obsahuje kompletnú realizáciu JMS konektoru. Pre implementáciu komponentu JMS konektor je využitý balík Java Message Service API, ktorý poskytuje spoločnú cestu pre vytváranie, odosielanie, prijímanie a čítanie

správ systému zasielania správ podniku **[31].**
Trieda implementuje rozhrania Connector a IJMSConnector. Rozhranie Connector predpisuje jednotnú formu pre všetky konektory knižnice a obsahuje metódy pre inicializáciu konektoru, nastavenie pripojenia, metódy pre nastavenie ďalších objektov a identifikátorov požadovaných systémom a metódu
pre ukončenie práce s konektorom. Konkrétne parametre pripojenia môžu byť používateľom jednoducho nastavené z webového rozhrania prostredia – Dashboard.
Implementácia rozhrania IJMSConnector vyžaduje, aby boli prítomné metódy na prijímanie správ a spracovanie viacerých správ za pomoci externého objektu, metóda na odosielanie textových správ, metóda na odosielanie objektov a metóda
na nastavenie maximálnej doby čakania na prijatie správy. Implementácia konektoru nie je závislá na tom či sa pripája do témy, lokality slúžiacej pre rozosielanie správ viacerým používateľom alebo fronty, ktorá správu zašle vždy iba jednému používateľovi.
JmsStatisticTool je trieda, ktorá zhromažďuje štatistické dáta o jednotlivých prenosoch správ vykonaných za pomoci JMS konektoru. Jednotlivé záznamy sú zbierané volaním metódy register(), ktorá ako svoje parametre berie názov destinácie, reťazec obsahujúci prvých 1000 znakov správy a reťazec značiaci typ operácie, ktorá bola vykonaná. V súčasnej dobe je podporovaný reťazec „send“ značiaci odchádzajúcu správu a reťazec „receive“, ktorý označuje správu prijímanú. Pre akýkoľvek iný reťazec metódy vypíše varovanie oznamujúce, že operácia nie je podporovaná a nebola zaznamenaná. Štatistické dáta sú do databázy ukladané pri každom zavolaní metódy.

4.3.9 Konektor pre spúšťanie rozhraní

Posledný konektor, ktorý slúži pre spúšťanie iných rozhraní počas behu pôvodného. Rozhraním sa myslí program bežiaci v prostredí Unified Environment, ktorého cieľom je presunúť dáta z jedného miesta na druhé, pričom ich zvyčajne modifikuje, prípadne validuje.
Trieda SynchInterfaceConnector obsahuje metódy poskytujúce funkcionalitu požadovanú po tomto konektore. Trieda implementuje rozhrania Connector a ISynchInterfaceConnector. Rozhranie Connector, tak ako vo všetkých predošlých konektoroch, obsahuje metódy pre inicializáciu konektoru, nastavenie pripojenia, metódy pre nastavenie ďalších objektov a identifikátorov požadovaných systémom a metódu pre ukončenie práce s konektorom. Konkrétne parametre pripojenia môžu byť opäť používateľom jednoducho nastavené z webového rozhrania prostredia – Dashboard.
Implementácia rozhrania ISyncInterfaceConnector vyžaduje, aby boli prítomné metódy na spustenie iného rozhrania, pričom sú mu predané spúšťacie parametre, a to buď s využitím informácií predaných konektoru v objekte pripojenie alebo pomocou textového reťazca. Je potrebné, aby bolo možné v tomto reťazci namapovať jednotlivé premenné buď z objektu, alebo mapy. Sú prítomné metódy pre volanie ako na prednastavenom rozhraní, tak aj na rozhraní poskytnutom za behu.

Štatistiky pre tento konektor nebolo nutné zaznamenávať, údaje o spúšťaní rozhraní sú zhromažďované na inom mieste systému.

4.4 Testovanie a nasadenie knižnice

Počas celej doby vývoja knižnice, boli jednotlivé komponenty testované za pomoci JUnit testov a tam, kde by bolo nasadenie jednotkových testov z hľadiska komplexnosti riešenia náročné, boli konektory otestované nasadením na k tomu účelu vytvorené rozhrania. Aby bola aplikácia nasadená v produkčnom prostredí spoločnosti ON Semiconductor, musí prejsť integračným (INT) a akceptačným (UAT) testovaním vykonávaným inými pracovnými tímami. Knižnica bola pred nasadením do ostrej prevádzky dostatočne a dôsledne otestovaná. Po nasadení bolo potvrdené splnenie požiadaviek, ktoré boli kladené na knižnicu pri zadaní práce.

Kapitola 5
Záver

5.1 Zhodnotenie práce a budúci vývoj
Cieľom tejto práce bolo vytvoriť aplikáciu, ktorá pristupuje k dátovým zdrojom a môže byť nasadená a využívaná v produkčnom prostredí spoločnosti ON Semiconductor.
Knižnica bola implementovanáa splnila všetky požiadavky, ktoré boli na jej vývoj kladené. Vývoj sa nezaobišiel bez chýb a zdržaní, problémy sa týkali najmä výberu správnych balíkov a frameworkov poskytujúcich všetky funkcie vyžadované pre vyriešenie pripojenia ku konkrétnemu dátovému zdroju.
Práca sprevádza celým systémom, v ktorom je táto aplikácia nasadená, popisuje základy architektúry orientovanej na služby a oboznamuje čitateľa s procesom vývoja a funkciami samotnej knižnice pre prístup k dátovým zdrojom.
Napriek tomu, že boli všetky definované požiadavky splnené, projekt má nepochybne ďalší potenciál k rozvoju a rozšíreniu poskytovaných služieb. Medzi možnosti ďalšieho vývoja projektu patrí pridanie ďalších konektorov k dátovým zdrojom, zlepšenie práce so štatistikami a prípadné ďalšie rozširovanie funkcií súčasných konektorov, potreba ktorého vzíde s rozširovaním systému a počtu rozhraní. Výsledkom práce je však funkčná aplikácia, v dobe písania práce (2015) plne postačujúca pre potreby spoločnosti ON Semiconductor.

Literatúra

[1]
Reference Architecture Foundation for Service Oriented Architecture Version 1.0.04 [online]. December 2012. OASIS Committee Specification 01 [cit. 2015-04-24]. Dostupné z: http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/cs01/soa-ra-v1.0-cs01.html

[2]
UDAYAKUMAR, K. *Oracle SOA infrastructure implementation certification handbook (1Z0-451): successfully ace the 1Z0-451 Oracle SOA Foundation Practitioner exam with this hands on certification guide : [professional expertise distilled]*. Birmingham [England] : Olton, vi, 354 p. ISBN 9781849683418.

[3]
BREEST, M. *An Introduction to the Enterprise Service Bus* [online]. Hasso-Plattner-Institute for IT Systems Engineering at the University of Potsdam, Potsdam, Germany, 2006 [cit. 2015-04-24]. Dostupné z: http://www.cin.ufpe.br/~in1062/intranet/bibliografia/fose-the_enterprise_service_bus.pdf

[4]
VALIPOUR, M.H. AMIRZAFARI, B. MALEKI, K.N. DANESHPOUR, N, *A brief survey of software architecture concepts and service oriented architecture.* Shahid Rajaee University, Tehran, Iran, ISBN: 978-1-4244-4519-6

[5]
ERL, T. *SOA: Principles of Service Design* [online]. 2008 [cit. 2015-08-28]. Dostupné z: http://serviceorientation.com/static/pdf/SOA_Principles_Poster.pdf

[6]
JOSUTTIS, N. *SOA in practice*. Farnham: O'Reilly. 2007
ISBN: 978-0-596-52955-0

[7]
ERL, T. *SOA Principles of Service Design*. Upper Saddle River, NJ : Prentice Hall. 2008
ISBN: 0-13-234482-3

[8]
ERL, T. *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*. Prentice Hall PTR, Upper Saddle River, NJ, USA 2004, ISBN: 007-6092025443

[9]
SCHMIDT, M.-T. HUTCHISO, B. LAMBROS, P. PHIPPEN, R. *The Enterprise Service Bus: Making service-oriented architecture real* IBM Systems Journal Issue, 2005 ISSN: 0018-8670

[10]
VIEIRA, M. COSTA, A.-C. MADEIRA, H. " Towards Timely ACID Transactions in DBMS", Dependable Computing, 2006. PRDC '06. 12th Pacific Rim International Symposium, ISBN: 0-7695-2724-8

[11]
BHAT, M. *SOA? ESB? What is all this* [online]. 2008 [cit. 2015-09-02] Dostupné z: https://software.intel.com/en-us/articles/soa-esb-what-is-all-this

[12]
SKONNARD, A. *A Developer's Guide to the Service Bus in Windows Azure AppFabric* Pluralsight, 2009

[13]
IBM: Model and build ESB SOA frameworks [online]. 2013, [cit. 2015-09-02] Dostupné z: https://soadevelopers.wordpress.com/2013/09/18/ibm-model-and-build-esb-soa-frameworks/

[14]
JBoss Portal Documentation Library [online]. 2009, [cit. 2015-09-02] Dostupné z: http://docs.jboss.com/jbportal/v2.7.1/userGuide/html/

[15]
DAVIS, J. *Open Source SOA*, Manning Publications, 2009, ISBN 978-1-933988-54-2

[16]
SOA registry [online]. [cit. 2015-09-02] Dostupné z: http://searchsoa.techtarget.com/definition/SOA-registry

[17]
ARPACI-DUSSEAU, R.-H. ARPACI-DUSSEAU, A.-C. *Operating Systems: Three Easy Pieces* Arpaci-Dusseau Books, 2015

[18]
LORENTZ, D. *Oracle Database SQL Language Reference* [online]. [cit. 2015-11-02] Dostupné z: https://docs.oracle.com/cd/B28359_01/server.111/b28286.pdf

[19]
RISBERG, T.*Spring Data JDBC Extensions Reference Documentation* [online]. [cit. 2015-11-02] Dostupné z: http://docs.spring.io/spring-data/jdbc/docs/current/reference/pdf/spring-data-jdbc-ext-reference.pdf

[20]
CHUNG, C. *An Introduction to FTP* [online]. [cit. 2015-11-02] Dostupné z: http://www.2brightsparks.com/resources/articles/an-introduction-to-ftp.pdf

[21]
FTPClient (Apache Commons Net 3.4 API) [online]. Commons.apache.org, 2015. [cit. 2015-11-02] Dostupné z: https://commons.apache.org/proper/commons-net/apidocs/org/apache/commons/net/ftp/FTPClient.html

[22]
CHUNG, C. *SSH File Transfer Protocol (SFTP) Explained* [online]. [cit. 2015-12-06] Dostupné z: http://www.2brightsparks.com/resources/articles/ssh-file-transfer-protocol-explained.pdf

[23]
STRACHAN, J. *Commons VFS - Commons Virtual File Systém* [online]. Commons.apache.org. 2015. [cit. 2015-12-06] Dostupné z: https://commons.apache.org/proper/commons-vfs/index.html

[24]
RIABOV, V. V. *SMTP (Simple Mail Transfer Protocol*. Rivier College [online]. [cit. 2015-12-06] Dostupné z: https://www.rivier.edu/faculty/vriabov/Information-Security-SMTP_c60_p01-23.pdf

[25]
JAVAMAIL API TUTORIAL [online]. tutorialspoint.com [cit. 2015-12-06] Dostupné z: http://www.tutorialspoint.com/javamail_api/javamail_api_tutorial.pdf

[26]
FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H., MASINTER, L., LEACH, P., BERNERS-LEE, T. *Hypertext Transfer Protocol -- HTTP/1.1* [online]. RFC 2616, 1999, [cit. 2015-12-28] Dostupné z: http://www.rfc-editor.org/info/rfc2616

[27]
Apache HttpComponents - HttpComponents HttpClient Overview [online]. Hc.apache.org, 2015. [cit. 2015-12-28] Dostupné z:
https://hc.apache.org/httpcomponents-client-4.5.x/index.html

[28]
Web Services Architecture [online]. W3.org, 2015. [cit. 2015-12-28] Dostupné z:
http://www.w3.org/TR/ws-arch/

[29]
javax.xml.soap (Java Platform SE 7 [online]. Docs.oracle.com, 2015[cit. 2015-12-28] Dostupné z:
http://docs.oracle.com/javase/7/docs/api/javax/xml/soap/package-summary.html

[30]
HAPNER, M., BURRIDGE, R., SHARMA, R., FIALLI, J., STOUT, K.
Java Message Service Specification [online]. Sun Microsystems, Palo Alto, California, 2002. [cit. 2015-12-28] Dostupné z:
http://download.oracle.com/otn-pub/jcp/7195-jms-1.1-fr-spec-oth-JSpec/jms-1_1-fr-spec.pdf

[31]
javax.jms (Java EE 6) [online]. Docs.oracle.com, 2015, [cit. 2015-12-28] Dostupné z:
http://docs.oracle.com/javaee/6/api/javax/jms/package-summary.html

Zoznam elektronických príloh

Archív záverečnej práce v IS MU obsahuje tieto elektronické prílohy:

- text práce vo formáte PDF
- archív so zdrojovými kódmi knižnice ConnectionProvider a pomocnými triedami