Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Komunikace s řídící jednotkou pro ovládání poloautomatických VNA vozíků

Plzeň 2017 Martin Kantořík

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 28. června 2017

Martin Kantořík

Poděkování

Děkuji společnosti Aimtec a. s. za možnost zpracovat tuto bakalářskou práci a vedoucímu práce Ing. Janu Brnkovi za vedení a dohled nad mojí prací. Dále bych chtěl poděkovat i mému konzultantovi na fakultě aplikovaných věd Dr. Ing. Karlovi Dudáčkovi za rady a připomínky.

Abstract

Bachelor thesis aims to ensure communication between the control unit and application for warehouse management in company. Commnunication is ensured by using TCP. In this work, we will approach the operation of the control unit in the warehouse and its requirements for the warehouse management. After that, the work will focus on DCIx architecture and discuss possible solutions. After choosing the solution, the thesis deals with its implementation. At last it will describe form of testing and creation of documentation within the DCIx application.

Abstrakt

Bakalářská práce má za cíl zajištění komunikace mezi řídící jednotkou a aplikací pro správu skladu ve firmě. Komunikace je zajištěna pomocí TCP. V této práci se přiblíží fungování řídící jednotky ve skladu a její požadavky na skladový systém. Dále se práce zaměří na architekturu aplikace DCIx a rozeberou se možná řešení. Po výběru řešení se práce zabývá jeho implementací. Nakonec se popíše způsob testování a vytvoření dokumentace v rámci aplikace DCIx.

Obsah

1 Úvod **1**

2 Řídící jednotka pro ovládání poloautomatický VNA vozíků **2**
2.1 Představení KBOXu 2
2.2 Poloautomatický vozík 2
2.3 VNA navigační systém 2
2.4 Omezení pro skladový systém 3
2.5 Specifikace KBOXu 3
2.6 Transmission Control Protocol 5
2.6.1 Příznaky 5
2.6.2 Hlavička TCP 6
2.6.3 Navázání spojení 6
2.6.4 Ukončení spojení 7
2.7 Zprávy v komunikaci s řídící jednotkou 7
2.8 Příkaz k jízdě 9

3 Informační systém DCIx **11**
3.1 Představení DCIx 11
3.1.1 EDI a ERP 11
3.2 Řízení skladů a materiálové logistiky 11
3.3 Systém pro řízení nakládky a vykládky 13
3.4 Systém pro podporu operativního plánování 13
3.5 Systém pro řízení kvality procesů 14
3.6 Portálová kooperace odběratelů s dodavateli 15
3.7 Řízení denních odvolávek a sekvenčních dodávek 16

4 Architektura DCIx **18**
4.1 Datová třída 19
4.1.1 Uložené procedury 20
4.1.2 Funkce 20
4.1.3 Pohledy 20
4.1.4 Spoušť 20
4.1.5 Vysvětlení pojmu SQL a T-SQL 20
4.1.6 Transakce, žurnály a návrat změn 21
4.1.7 Relační databáze 22
4.2 Byznys třída 23

4.2.1 Transakce v DCIx 23
4.2.2 Final Values 24
4.2.3 Moduly 24
4.3 Webová třída 24
4.3.1 Telnet 24
4.3.2 JavaServer Pages 24

5 Návrh logiky komunikace **25**
5.1 Použitelnost řídící jednotky ve skladu 25
5.2 Uvažování nad funkcemi DCIx 25
5.3 Uvažování nad modulem 26
5.3.1 Modul sendMessageToListener 26
5.4 Kompatibilita souřadnic skladu 27
5.5 Použitelnost DCIx ve skladu 28
5.6 Shrnutí návrhu 28

6 Implementace **29**
6.1 Rozšíření databázových tabulek 29
6.2 Vytvoření KBOX posluchače 30
6.2.1 Přijímání zpráv 31
6.2.2 Odesílání zpráv 32
6.3 Vytvoření procedur 33
6.4 Problémy během vývoje 34

7 Testování **35**
7.1 Testovací transakce 35
7.2 Jednotkové testy 35
7.2.1 Testování příchozích zpráv 35
7.2.2 Testování získání správného typu zprávy 36
7.2.3 Testování odchozích zpráv 36
7.3 Testování pomocí Selenium 36

8 Dokumentace **37**

9 Závěr **39**

Literatura **40**

1 Úvod

Práce vzniká pro společnost Aimtec a. s. se sídlem v Plzni. Cílem práce je navrhnout a implementovat způsob komunikování mezi řídící jednotkou pro ovládání poloautomatických VNA (*Very Narrow Aisle*) vozíků a produktem DCIx, který se soustředí na komplexní firemní logistické řešení. Řídící jednotka je nazývána KBOX a je vyráběna společností STILL s.r.o. k jejím poloautomatickým vozíkům.

První kapitola práce je zaměřená na představení zmíněné řídící jednotky. To zahrnuje specifikace dané jednotky, způsob jak se s tímto vybavením ve skladu může pracovat a nastíním i omezení, které jednotka má pro vybraný skladový systém. Posléze přibližuji komunikaci pomocí bezdrátové sítě a zprávy, které jednotka může zpracovávat. To znamená, jaký formát musí zprávy splňovat a typy zpráv, které je zařízení schopno přijímat a odesílat.

V druhé kapitole představuji samotný produkt DCIx. Zaměřuji se zde na pět jednotlivých oblastí celého produktu, abych pokryl jeho kompletní funkčnost. U každé oblasti stručně popisuji, co jednotlivá oblast nabízí, co tím zákazník získá a jak tyto oblasti mohou fungovat v praxi.

V následující kapitole se zaměřuji již na technické řešení celého produktu, abych získal povědomí o tom, jak celý produkt funguje a co všechno nabízí za použité technologie. Rozebírám zde jednotlivé části a následně jejich propojení i s následným výsledkem na uživatelském monitoru.

Další kapitola je zamyšlení nad konečným řešením mojí implementace komunikace s řídící jednotkou. Rozebírám zde několik způsobů, jak by dané řešení mohlo vypadat a nakonec odůvodním, proč jsem zvolil právě toto řešení.

Poslední kapitoly se věnují implementaci, testování a dokumentaci. U implementace je popisován UML model řešení a stručné vylíčení některých problémů, které se vyskytly v průběhu implementace. V neposlední řadě popíši i způsob testování vytvořeného řešení, který probíhal pomocí JUnit testů a nástrojem pro automatické testování webových aplikací nazývající se Selenium. Nakonec shrnu tvorbu dokumentace pro provedené řešení.

2 Řídící jednotka pro ovládání poloautomatický VNA vozíků

2.1 Představení KBOXu

Řídící jednotka KBOX je vyráběna a dodávána firmou STILL. K zařízení je často dodáván speciální VNA poloautomatický vozík, který je připraven na propojení s tímto zařízením. Vozík navíc obsahuje senzory pro zjišťování aktuální polohy ve skladu. Pro kompletní fungování firma dodává i speciální sklad, tedy regály s velice úzkými uličkami, které jsou uzpůsobeny pro maximální využití zmíněného vozíku. Každý regál totiž obsahuje kódy, podle kterých senzory umístěné na vozíku zjistí jeho přesnou polohu. Vozík se proto může jednoduše navigovat po celém skladu. KBOX samotný musí být pomocí bezdrátové sítě připojen k firemnímu skladovému systému, který musí s tímto zařízením umět komunikovat. Nelze tak připojit zařízení k jakémukoliv systému pro řízení skladu.

2.2 Poloautomatický vozík

Poloautomatickým vozíkem v této práci je nutno rozumět vysokozdvižný vozík, který má na předním panelu umístěn KBOX. Tento vozík musí řídit operátor, který bude dostávat pokyny pomocí tohoto přístroje. Ten hlídá, jestli jede správným směrem a zda dojel na správné místo. Vozík se sám od sebe nerozjede. Pro koordinaci vozíku po skladu se používá VNA navigační systém.

2.3 VNA navigační systém

VNA navigační systém je samostatně – v rámci systémových omezení – konfigurovatelný a standardizovaný asistenční systém pro VNA vozíky napomáhající operátorovi v navigování skladovými uličkami. V systému je sklad definovaný podle skladových oblastí, řad, sloupců, úrovní a pozic. K navigování mezi uličkami vozík potřebuje znát fyzické nominální souřadnice

cílové pozice. Fyzické souřadnice (XYZ) definují třídimenzionální souřadnicový systém uvnitř uliček [14].

2.4 Omezení pro skladový systém

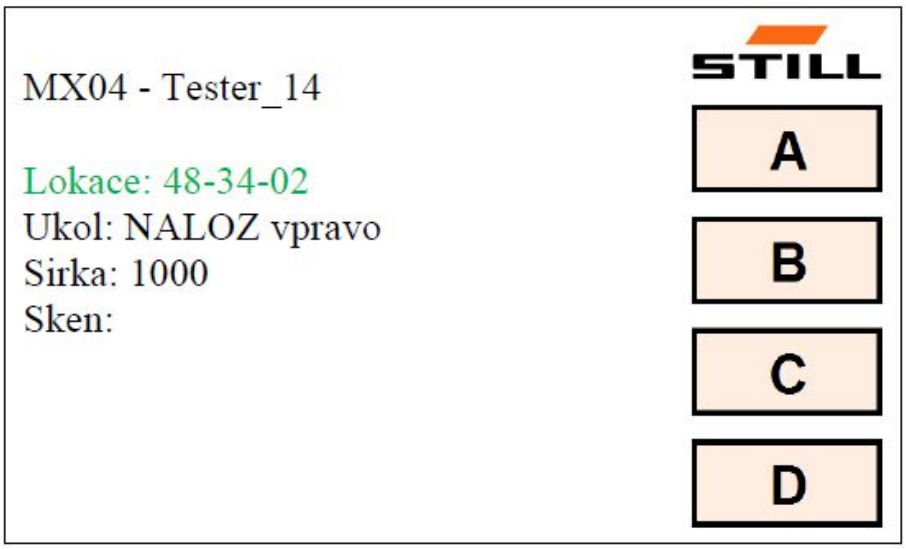
Pravidla, která musí skladový systém dodržovat, vycházejí z kapacity paměti v ovladači vozíku a použitým měřícím systémem. Měřící systém se dělí na dva typy. Prvním typem je měření podle čárových kódů. Druhým je měření podle RFID neboli identifikace na rádiové frekvenci. Některá omezení jsou společná pro oba typy. Oba systémy musí mít maximálně deset skladovacích oblastí ve skladu a maximálně 99 sloupců nebo maximálně 400 pozic pro jednu řadu. Jediným rozdílem je, že první typ může mít maximálně 168 úrovní a maximálně 500 uliček na jeden sklad a druhý typ má také maximálně 168 úrovní, ale může mít jen 255 uliček na jeden sklad.

Pozici ve skladu lze počítat dvěma způsoby. Buď použijeme inkrementační počítání, nebo segmentační počítání. Inkrementační počítání znamená, že počítáme jen podle pozic. Nepočítají se tedy žádné sloupce, ale každá pozice musí použít jedinečnou identifikaci v rámci řady. Při použití segmentovaného způsobu se počítá podle sloupce i podle pozice. Tento způsob musí splňovat dvě podmínky. Každý sloupec musí mít unikátní identifikaci v rámci řady a každá pozice musí použít unikátní identifikaci v rámci sloupce.

2.5 Specifikace KBOXu

Zařízení je osazeno dvěma USB porty, které mohou sloužit pro připojení čteček čárových kódů nebo se do nich zapojují USB zařízení nejčastěji pro zapsání logů ze zařízení či pro počáteční nastavení bezdrátového připojení. Jedním portem RS232 a modulem pro WLAN (*Wireless Local Area Network*). Pro zobrazování informací o vozíku a provádění jednoduchých akcí slouží sedmipalcový dotykový displej (viz Obrázek 2.1). Pokud je zařízení propojeno se skladovým systémem, objeví se na pravé straně dotykového displeje čtyři tlačítka označená velkými písmeny A, B, C a D. Tato tlačítka slouží pro komunikaci s operátorem, který pomocí těchto tlačítek může komunikovat se skladovým systém a provádět tak některé naprogramované akce. Funkce těchto tlačítek jsou zcela závislé na funkcionalitě daného systému, jelikož KBOX odesílá jen informaci o tom, které z tlačítek bylo právě stisknuto. Na displeji je místo pro deset řádek, na kterých můžeme vypisovat informace o úkolech a stavu skladu. Na každé řádce je navíc umožněno nastavit jinou barvu textu. V diagnostickém režimu se na desátém řádku

zobrazují diagnostické údaje, ze kterých lze vyčíst mnoho informací o síle signálu, odesílaných a přijímaných paketech. Na displeji lze zobrazit aktuální IP adresu pro snadnější nastavení připojení ze skladového systému.



Obrázek 2.1: Ilustrační obrázek zařízení KBOX [13]

Prostřednictvím bezdrátového připojení zařízení komunikuje se skladovým systémem za pomoci TCP (*Transmission Control Protocol*). Při tomto spojení se přístroj chová jako server, ke kterému se skladový systém připojuje, avšak toto zařízení podporuje pouze jediného připojeného klienta. Při pokusu o připojení nového klienta je stávající připojení ukončeno a posléze nahrazeno novým připojením s novým klientem. Tímto způsobem to funguje proto, aby nevznikal chaos v příchozích datech na KBOX, kdy by všichni klienti mohli vozík žádat o vyzvednutí jiného zboží a operátor u vozíku by poté těžce optimalizoval cestu, aby vozík nemusel jet mnohokrát stejnou trasu.

Řídící jednotka při přijímání zpráv od skladového systému vždy nejprve obdrženou zprávu zpracuje a poté odesílá zpět potvrzení o proběhlém zpracování. Může se tak zjistit, zda zpráva byla poslána korektně, tedy ve správném formátu. Pokud byla data korektní, dostaneme hned i odpověď s informacemi, o které jsme žádali či jen jednoduché potvrzení o přijetí dat, ve kterém se odesílají přijatá data jen s odlišným identifikačním znakem. Skladový systém tak může relativně snadno reagovat na stávající situaci.

Pro aktuální stav KBOXu se na zařízení nachází tři stavové diody. První dioda svítí zeleně, pokud je zařízení aktuálně zapnuté a je napájené ze sítě. Druhá dioda svítí oranžovou barvou, pokud je zařízení připojené k bezdrátové síti. Když tato dioda bliká, znamená to, že zařízení není připojeno.

Poslední je červená dioda, signalizující poruchu v zařízení.

Zařízení umožňuje ukládat kompletní záznam uskutečněné komunikace. Tento záznam je evidovaný ve vnitřní paměti, a proto je záznam dostupný i po restartu zařízení. Logovací soubor je textový, řádkový a inkrementální. Maximální velikost souboru je 200MB, pokud se tato velikost překročí, dojde ke smazání souboru a logování začíná znovu do nového prázdného souboru.

2.6 Transmission Control Protocol

Řídící jednotka komunikuje se systémem pro řízení skladu prostřednictvím TCP, proto je dobré vědět, jak tento protokol funguje. TCP vznikl v roce 1974 a poté byl ještě mnohokrát vylepšen. Jedná se o spojovanou transportní službu. Pojem spojovaná služba znamená, že před přenosem dat musí být navázáno spojení mezi příjemcem a odesílatelem. To nám zajišťuje doručení dat v nezměněném pořadí. Slovo *transportní* zde znamená, že služba operuje na transportní vrstvě modelu OSI, ta umožňuje adresovat přímo jednotlivé klienty pomocí jedinečných čísel portů. Čísla portů jsou uváděna jako 16bitová bezznaménková čísla. Každá komunikační strana má u sebe přiřazen port, přes který je uskutečněno navázané spojení.

Navíc se jedná o protokol se spolehlivým doručováním, takže se TCP stará o správné odesílání paketů či opakované odeslání v případě ztracení nebo poškození některého z paketu. K tomuto zabezpečení používá protokol potvrzovací zprávy, které jsou odeslány po úspěšném obdržení zprávy. Pokud potvrzení nedorazí v určeném časovém intervalu zpět k odesílateli, odešle se nepotvrzená zpráva znovu. Poškození paketu je hlídáno pomocí kontrolního součtu, který je odesílán se zprávou k příjemci. Příjemce poté ze zprávy vypočte svůj kontrolní součet, pokud se nakonec tyto součty neshodují, je paket zahozen a čeká se na nový.

Každá aplikace předává protokolu celou zprávu v bajtech a ten si poté zprávu sám rozdělí do přiměřeně velkých segmentů. Velikost těchto segmentů se určuje parametrem MTU (*maximální přenosová jednotka*) na linkové vrstvě u daného připojeného počítače. Tato hodnota se nejčastěji pohybuje okolo 1500 bajtů. Po tomto rozdělení může posílat zprávu příjemci.

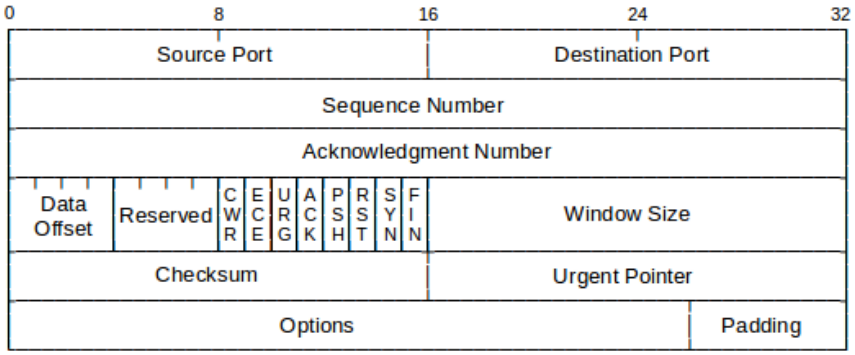
2.6.1 Příznaky

Protokol používá aktuálně osm příznaků, které jsou uloženy na šesti bitech. Jsou jimi URG (*Urgent*), ACK (*Acknowledge*), PUSH, RST (*Reset*), SYN (*Synchronize*), ECE (*Explicit congestion notification - Echo*), CWR (*Congestion Window Reduced*) a posledním je FIN. FIN je využíván pro ukončo-

vání navázaného spojení mezi serverem a klientem. Příznak SYN je naopak využíván v navazování spojení. Příznak URG se používá pro data, která mají vyšší důležitost než ostatní a jsou proto zpracovávaná přednostněji. PUSH se používá, pokud chceme poslat všechna neodeslaná data, může se stát, že protokol bude čekat na nová data, aby se naplnil segment. RST se používá pro resetování navázaného spojení. ECE slouží jako signalizační mechanismus pro předcházení zahlcení a CWR s ním úzce souvisí, protože vyjadřuje snížení velikosti okénka pro zahlcení [1]. Posledním příznakem je ACK, který se používá pro potvrzení obdržené zprávy.

2.6.2 Hlavička TCP

Pole v hlavičce TCP (viz Obrázek 2.2) určují zdrojový a cílový port, hodnoty sekvence, potvrzení a příznaky, podle kterých se zařízení řídí při zpracování obsahu paketu. Délka hlavičky je obvykle 20 bajtů, pokud nejsou použity žádné volby. Hlavička nemusí obsahovat žádné volby ani data [11].



Obrázek 2.2: Znázorněna hlavička paketu TCP [8]

2.6.3 Navázání spojení

Navázání spojení u tohoto protokolu probíhá vždy ve třech krocích. Prvním krokem je, že klient odešle datagram s nastaveným příznakem SYN, náhodně vygenerovaným číslem sekvence a potvrzovacím číslem nastaveným na nulu. Druhým krokem je, že server odešle klientovi další datagram s příznaky SYN a ACK, upraví potvrzovací číslo tak, že přičte k přijatému číslu sekvence jedničku a nakonec přidá do zprávy nově náhodně vygenerované číslo sekvence. Posledním krokem je odeslání datagramu od klienta s nastaveným příznakem ACK, číslo sekvence je upraveno, tak že je rovno vygenerovanému číslu klientem v prvním kroku a přičtením jedničky. Ke zprávě pak přidá ještě

číslo odpovědi, která je vyjádřena přičtenou jedničkou k obdrženému číslu sekvence od serveru. Po obdržení zprávy serverem je spojení navázáno. Obě strany si tak pamatují obě čísla sekvencí, která jsou později použita pro určování pořadí paketů v komunikaci. Spojení zůstane navázáno do provedení ukončení spojení.

2.6.4 Ukončení spojení

Ukončení spojení probíhá ve čtyřech krocích na rozdíl od navázání. V prvním kroku klient odešle datagram s nastaveným příznakem FIN pro oznámení, že se klient odpojuje. Druhý krok je tvořen odpovědí serveru s nastaveným příznakem ACK. Ve třetím kroku i server odešle zprávu s nastaveným příznakem FIN a v posledním kroku klient odešle potvrzení o přijetí ukončení spojení od serveru. Po proběhnutí těchto čtyř kroků je spojení ukončeno.

2.7 Zprávy v komunikaci s řídící jednotkou

Pro komunikaci s řídící jednotkou musí zprávy splňovat formát zpráv, které umí zařízení zpracovávat. Formát těchto zpráv je dán tak, že zpráva musí začínat znakem STX (*Start of Text*, hexadecimální hodnota v ASCII tabulce je 02) a musí končit znakem ETX (*End of Text*, hexadecimální hodnota je 03). Mezi těmito znaky se nachází obsah zpráv posílaných do zařízení a ze zařízení.

Komunikace se může dělit na dva typy zpráv. Prvním typem jsou zprávy, které nemusí obsahovat svoji délku na začátku zprávy. Do tohoto typu patří zprávy, které zařizují výpis textu na displej, signalizace stisknutí tlačítek a v neposlední řadě odesílání naskenovaného čárového kódu. U těchto zpráv je na prvním místě vždy znak identifikující rozhraní, které tuto zprávu má přijmout či ze kterého jsou data odesílána.

Celkem lze identifikovat pět rozhraní. Prvním je displej identifikující pomocí písmene *D*. Zprávu s tímto identifikátorem lze odeslat jen ze skladového systému na zařízení. Při zobrazování textu můžeme libovolně volit barvu textu na řádce pro zpřehlednění informací. Druhým je USB port označovaný písmenem *U*. S tímto identifikátorem nám mohou chodit zprávy ze čtečky čárových kódů. Při skenování se čtečka chová jako US klávesnice a pro ukončení skenování je nutno zadat dvojici znaku CR a LF. Pokud tyto znaky nepřijdou, budu se data ze čtečky ukládat do bufferu dál. Po příchodu ukončovací sekvence odešle KBOX naskenovaná data bez těchto dvou znaků. Dalším rozhraním je port RS 232 značený písmenem *R*. Tento port

slouží ke komunikaci s vozíkem a zjišťování jeho aktuální polohy. Předposledním rozhraním je dotyková klávesnice, která je značena pomocí velkého písmene K. Tyto zprávy slouží k předání informace, o tom jaké tlačítko bylo právě stisknuto. Vyskytují se zde pouze čtyři zprávy a to s obsahem A, B, C nebo D podle stisknutého tlačítka. Posledním rozhraním je správa času s identifikátorem T pro nastavení času a znakem t pro zjištění času.

Do těchto rozhraní kromě jednoho pro zjišťování času, mohou být data složena z libovolných ASCII znaků kromě STX a ETX, zařízení by totiž bralo znaky jako označení konce nebo začátku zprávy a podle toho by se zachovalo. Pokud se vytváří zpráva pro manipulaci s časem, musí znaky být jen čísla ve formátu YYMMDDHHNNSS.

Druhým typem jsou zprávy, které musí obsahovat celkovou délku zprávy. Tyto zprávy slouží přímo k zadávání příkazu a získávání informací z vysokozdvižného vozíku. Tento typ lze ještě rozdělit na zprávy odesílané z vozíku a data odesílaná z řídícího systému. Rozeznají se od sebe velice snadno, protože zprávy odeslané z vozíku jsou vždy identifikovány malými písmeny a z řídícího systému písmeny velkými. U těchto zpráv existuje omezení na jejich obsah. Opět platí, že se znaky STX a ETX mohou vyskytovat pouze na začátku a konci zprávy. Navíc zde se zadává více informací jako například pozice ve skladu, a proto se používá středník jako oddělení těchto informací. Středník tedy musí být použit jen jako oddělení těchto informací a nesmí se vyskytovat v žádném názvu.

Existují celkem čtyři typy zpráv odesílaných z řídícího systému na vozík. Prvním je příkaz k jízdě. V této zprávě lze nalézt informace o tom, zda máme zboží vyzvednout z regálu či do něj uložit. Poté jsou zde souřadnice skládající se ze skladové oblasti, řady, sloupce, úrovně a pozice. K tomuto popisu se přidává ještě číslo horizontálního posunutí vidlic, pokud ovšem toto číslo zadané není, tak se bere posunutí o hodnotě nula. Druhým typem je resetovací zpráva, která smaže všechny příkazy a akce právě prováděné vozíkem. Předposledním je příkazová zpráva sloužící jako požadavek o stavu vozíku. Posledním je potvrzovací zpráva zajišťující potvrzení přijetí zprávy z vozíku.

U druhého typu zpráv, tedy zprávy posílané z vozíku do skladového systému, existuje pět typů zpráv. Prvním je potvrzení pozice, zasílané vždy, když vozík dorazil na horizontální i vertikální určenou pozici. Druhým je potvrzení příkazu. Tato zpráva slouží pro informaci, že vozík dokončil vyzvednutí či uložení balíčku na určenou pozici. Třetím je potvrzovací zpráva, která pouze posílá ACK nebo NAK v případě špatné interpretace zprávy. V předposlední zprávě jsou údaje o stavu vozíku. Mezi těmito daty lze nalézt aktuální pozici vozíku ve skladu, chybné hlášky v případě vyskytující

se chyby a jaký příkaz je právě prováděn. Posledním je zpráva obsahující operační data a stav baterie. Tato zpráva je automaticky odesílána každých 300 vteřin či při restartování zařízení. Může být také vyžádána pomocí vyslání požadavku na řídící jednotku.

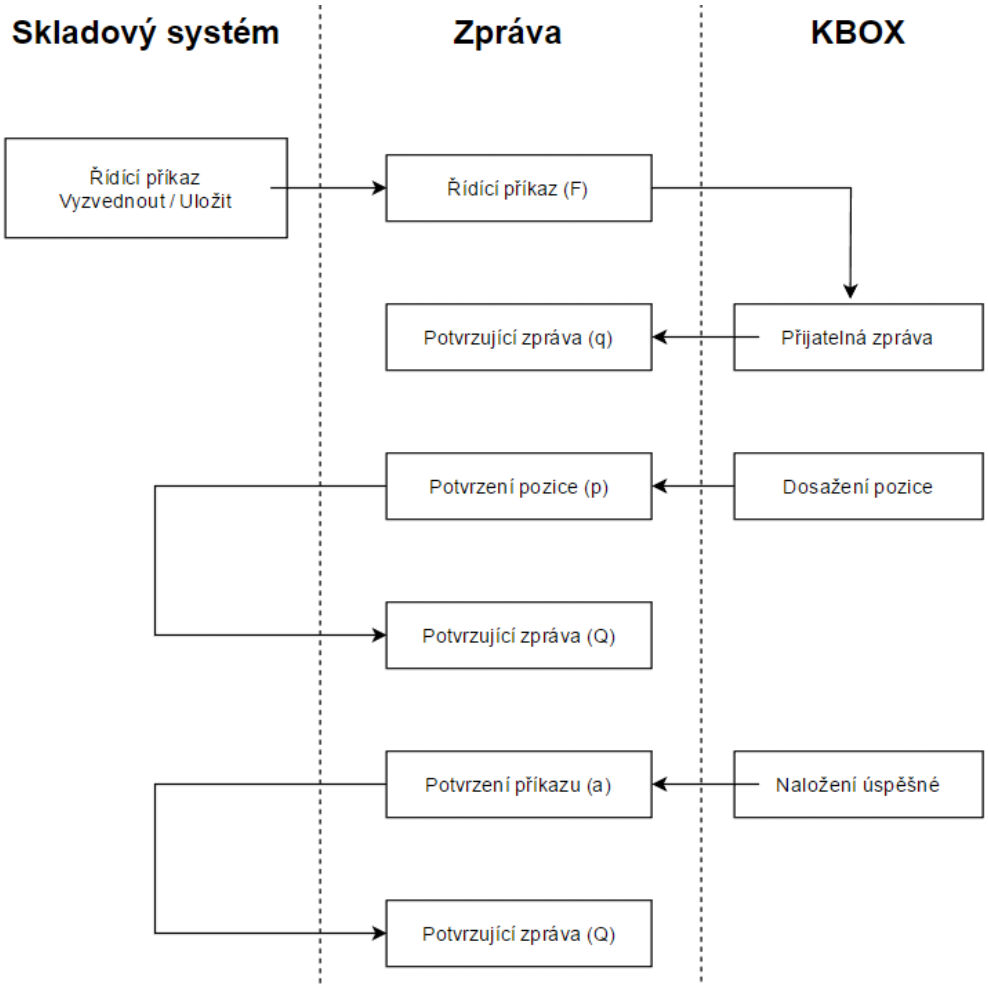
2.8 Příkaz k jízdě

Příkaz k jízdě patří k těm nejdůležitějším zprávám, které lze posílat ze skladového systému na vozík a je to hlavní funkcionalita, bez které by celý tento systém neměl smysl. Právě proto se musí dobře prozkoumat. Vozík podporuje tři příkazy k jízdě – vyzvednutí, uložení a zastavení pro manuální vyzvednutí.

Vyzvednutí balíčku je provedeno ve čtyřech krocích. Vozík nejdříve zkontroluje, zda se mu na vidlicích nenachází žádný náklad. Pokud jsou vidlice prázdné, může se přesunout na určenou destinaci, jakmile dorazí na místo, nastaví vidlice přímo proti cílovému balíčku. V této chvíli se pošle potvrzení pozice. Když je balíček v pořádku vyzvednutý z určené pozice, odešle se potvrzení s tím, že je příkaz dokončen a je očekáván nový úkol. Jak vypadá komunikace mezi vozíkem a skladovým systémem je vyobrazeno na obrázku 2.3. Na obrázku lze vidět, že skladový systém nejdříve pošle příkaz k vyzvednutí zásilky. Vozík po zkontrolování vidlic a korektnosti zprávy odešle potvrzovací zprávu. Jakmile vozík dosáhne požadované pozice, pošle další potvrzovací zprávu tentokrát s tím, že dojel na určenou pozici. Skladový systém může opět odpovědět potvrzením. Po úspěšném naložení zásilky na vozík přijde poslední potvrzení o splnění příkazu. Nakonec systém může opět odpovědět potvrzením.

Uložení balíčku je téměř stejné jako předchozí vyzvednutí s tím rozdílem, že na začátku vozík zkontroluje naloženou paletu na vidlicích a posléze ji uloží na určené místo.

Zastavení pro manuální vyzvednutí je však trochu odlišné. Status vidlic se nezjišťuje, takže vidlice mohou být naložené nebo úplně prázdné. Poté vozík odjede na danou pozici s tím, že zarovná tentokrát kabinu k pozici se zbožím. Jakmile vozík dosáhne pozice, je příkaz ohlášen jako splněný a pošle se potvrzovací zpráva.



Obrázek 2.3: Výměna zpráv mezi KBOXem a skladovým systémem

3 Informační systém DCIx

3.1 Představení DCIx

Produkt DCIx je jedním z produktů, které společnost Aimtec a. s. prodává. Jedná se o webovou aplikaci, která je zaměřená na řízení logistiky ve firmách. Řadí se mezi systémy kategorie MOM (*Manfacturing Operations Management*), který nejužším způsobem integruje firemní procesy ve výrobě a logistice. Při tomto pojetí využívá myšlenky Průmysl 4.0. Produkt obsahuje několik oblastí, kterými se zabývá. Tyto oblasti jsou následující: WMS (*Warehouse Management System*), PPS (*Production Planning System*), QMS (*Quality Management System*), PORTAL, YMS(*Yard Management System*), JIS (*Just In Sequence*) a JIT (*Just In Time*). DCIx integruje celý dodavatelsko-odběratelský řetězec, usnadňuje koordinaci a obchodní spolupráce se zákazníky, dodavateli a partnery. Zároveň může pouze doplňovat již existující zákaznické systémy o nové funkce, které jejich systém nepodporuje. Lze ho proto integrovat s různými typy informačních systémů například ERP (*Enterprise Resource Planning*) nebo EDI (*Electronic Data Interchange*)[2].

3.1.1 EDI a ERP

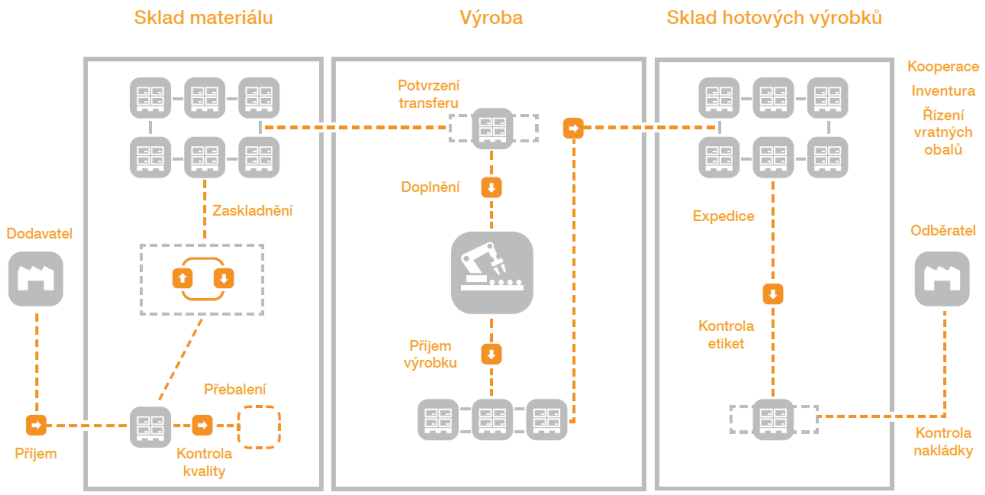
EDI slouží k elektronické výměně strukturovaných dat mezi počítačovými aplikacemi. Struktura dat je předem specifikována podle dohodnutých standardů a posléze mohou být data automaticky přenášena a zpracovávaná bez zásahu člověka.

Systém ERP slouží k integrování většiny oblastí podnikových činností, mezi které může patřit plánování výroby, marketing, výroba nebo správa materiálu.

3.2 Řízení skladů a materiálové logistiky

Nejdůležitější oblastí pro tuto práci je WMS (*Warehouse Management System*). Tato oblast rozšiřuje funkcionalitu podnikových informačních systémů o detailní řízení logistických procesů ve skladech a ve výrobě od příjmu až po expedici. Přináší spolehlivý a přesný zdroj informací v reálném čase s využitím identifikace manipulačních jednotek a skladových míst, pomocí

čárových nebo RFID kódů. Umožňuje řízení skladových operací, efektivnější využití stávajícího skladového prostoru nebo posílení produktivity pracovníků. Mezi podporované procesy patří přebalení zboží (homogenizace, mixované palety, kartony), vstupní a výstupní kontrola kvality materiálu, příjem na sklad, zaskladnění materiálu a mnoho dalších. Mezi největší výhody zavedení této oblasti patří zrychlení skladových procesů tím, že se odstraní hledání či chybovost. Dodržování pravidel FIFO (*First In First Out*) a FEFO (*First Expired First Out*) a kontrola přípravy palet a dalších. Metoda FIFO znamená, že první položku, která přijde do skladu, tak první ze skladu vyexpedujeme a v metodě FEFO expedujeme první tu položku, která má nejblíže k datu spotřeby.

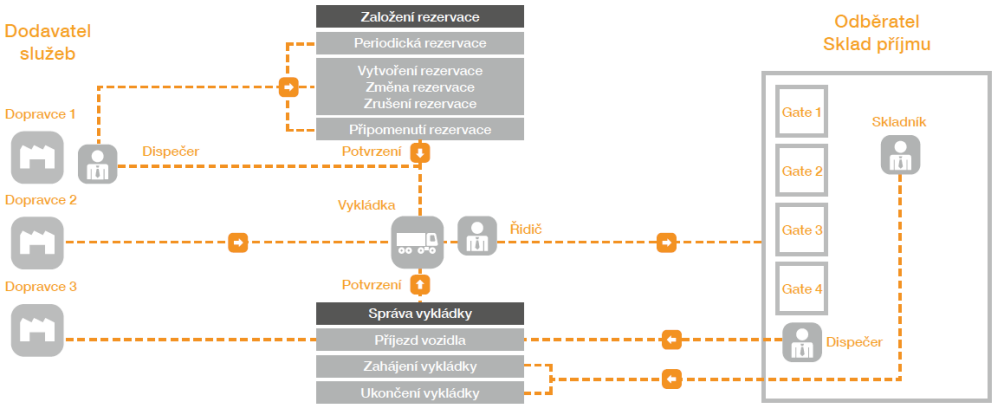


Obrázek 3.1: Ilustrační zobrazení fungování WMS [6]

Na obrázku 3.1 je blíže přiblížena funkcionalita WMS v DCIx, kdy dodavatel dodá materiál na příjem, po příjmu se materiál může nebo nemusí nechávat zadržet na kontrolu kvality a následné přebalení na lépe manipulovatelnou jednotku. Zaskladníme a poté můžeme předat materiál do haly s výrobou. Zde se materiál doplní do výroby a poté z linky vyjede hotový výrobek. Výrobek projde příjmem, kde se mu přiřadí etikety pro expedici a pokračuje dále na sklad, kde je následně expedován k odběrateli. Před naložením do kamionu jsou ještě zkontrolovány etikety, zda se expeduje opravdu správný výrobek a následně je výrobek odeslán k odběrateli.

3.3 Systém pro řízení nakládky a vykládky

Oblast YMS (*Yard Management System*) je grafická plánovací tabule přístupná přes internet dodavatelům, zákazníkům a případně dopravcům. Umožňuje hledat volné časové okno a následně v něm vytvořit rezervaci pro naložení či vyložení kamionu. Každá rezervace podstupuje automatickou validaci. Eviduje se čas začátku a konce vykládky. Tyto informace vedou k přesnějšímu plánování vlastních zdrojů. Tato oblast podporuje ještě následující procesy kromě již zmíněných a to tvorbu periodických rezervací, záznam historie nakládky a vykládky a další. Hlavní výhodou je snížení nákladů na organizaci nakládání a vykládání, zkrácení čekací doby dopravců a kapacitní plánování lidí a manipulačních jednotek.



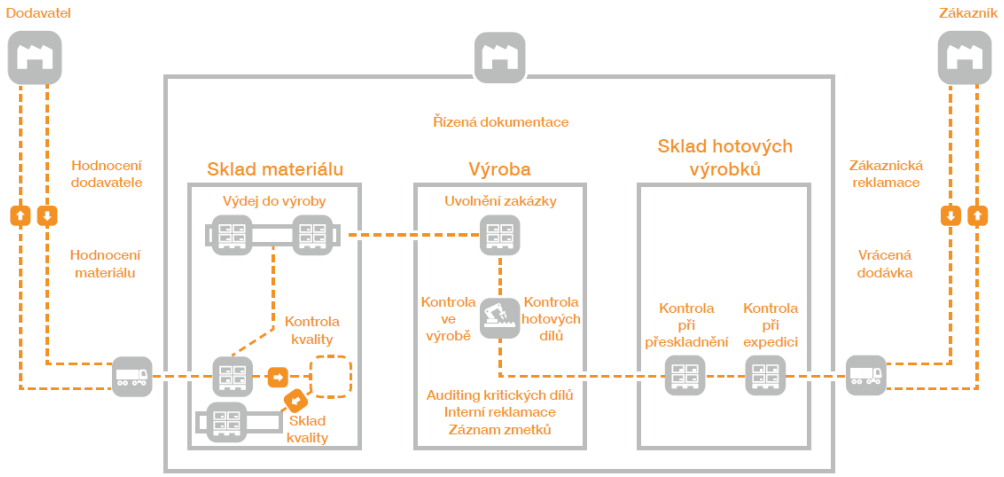
Obrázek 3.2: Ilustrační zobrazení fungování YMS [7]

3.4 Systém pro podporu operativního plánování

Oblast PPS (*Production Planning System*) představuje grafickou plánovací tabuli znázorňující výrobní zakázky alokované v čase na jednotlivé výrobní stroje, linky a pracoviště. Jsou zde zobrazované importované výrobní zakázky, vytvořené na základě MRP plánování, nebo systém dovoluje zadávat zakázky ručně. Pomocí tohoto modulu lze přesunovat zakázku v čase, přesunovat zakázku mezi zdroji, odstranění mezer v plánu nebo definovat výrobní kalendář pro každý stroj. Tím se docílí zlepšení využití zdrojů, zrychlení a zjednodušení operativního plánování výroby.

3.5 Systém pro řízení kvality procesů

Další oblastí je QMS (*Quality Management System*), to je systém pro řízení procesů kvality. Jeho účelem je řízení vstupní a výstupní kontroly, monitoring a kontrola dodržování stanovených pravidel nastavených oddělením kvality. Dále obsahuje funkce jako kontrola kvality na úrovni šarží, manipulačních jednotek nebo kusů. Obsahuje propracovaný systém hodnocení dodavatelů a pracuje s dynamickými plány kontroly a hodnocením procesů výroby. QMS dále podporuje velké množství procesů prováděných v podniku. Mezi tyto procesy patří příjmy a výdaje z blokačního skladu, evidence spotřeby vzorků na destruktivní testy, zadržení a uvolnění kvalitou nebo rozdělení dodavatelů do různých kategorií podle kvality dodávek. Po zavedení tohoto systému by mělo dojít ke snížení nákladů na řízení kontroly materiálu rozpracované výroby a finálních produktů, zlepšení úrovně kvality přejímaných dílů a identifikace neshodných dílů na úroveň manipulačních jednotek.

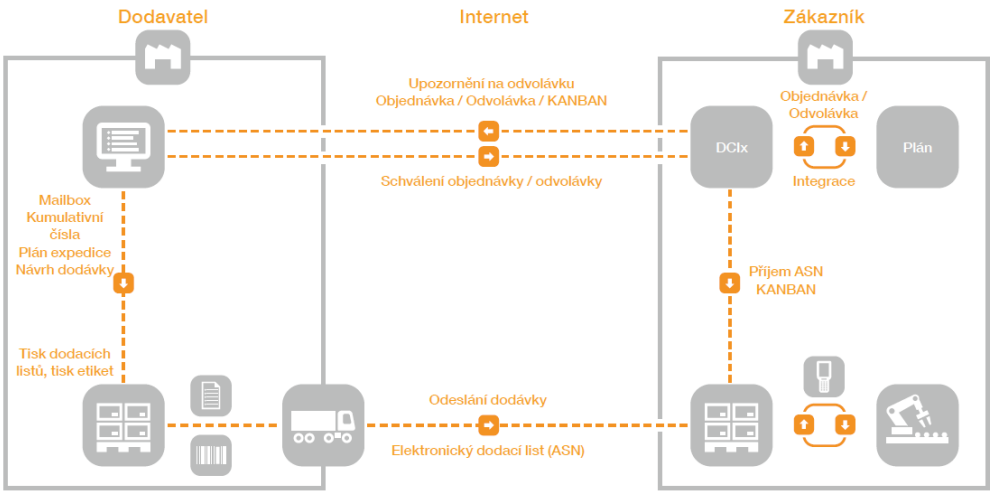


Obrázek 3.3: Ilustrační zobrazení fungování QMS [5]

Na obrázku 3.3 lze vidět, že z leva se hodnotí jak dodavatel, tak dodávaný materiál. Poté se ve skladu kontroluje kvalita. Ve výrobě se může kontrolovat kvalita během výroby a potom i hotový výrobek. Na skladu hotových výrobků se provádí dvojitá kontrola, když přeskladňujeme výrobky a nakonec když je expedujeme. Ze strany zákazníka se evidují reklamace a jaké dodávky s výrobky byly vráceny.

3.6 Portálová kooperace odběratelů s dodavateli

Předposlední oblast se nazývá PORTAL a slouží k usnadnění spolupráce mezi dodavatelem a odběratelem. Obsahuje ucelený přehled o aktuálně schválených objednávkách, odvolávkách, stavech skladů a pohybu zásob. Mezi hlavní funkce systému patří komunikace s dodavateli přes internet a poskytuje již několik předkonfigurovaných procesů pro různé dodavatelské scénáře, jakými jsou například objednávky či zavedení konsignačního skladu. Konsignační sklad je takový sklad, ve kterém se skladuje zboží nepatřící vlastníkovi skladu. Nejčastěji se využívá za účelem přiblížení zboží k zákazníkům. Celkově tak podporuje firemní procesy týkající se například potvrzení objednávky dodavatele, hodnocení dodavatelů na základě přesnosti dodávek nebo usnadňuje tisk zákaznických etiket a dodacího listu. Zavedení vede k přesnějším informacím pro plánování nákupu a výroby, ale především i ke snížení pracnosti při objednávání a komunikace s dodavateli.

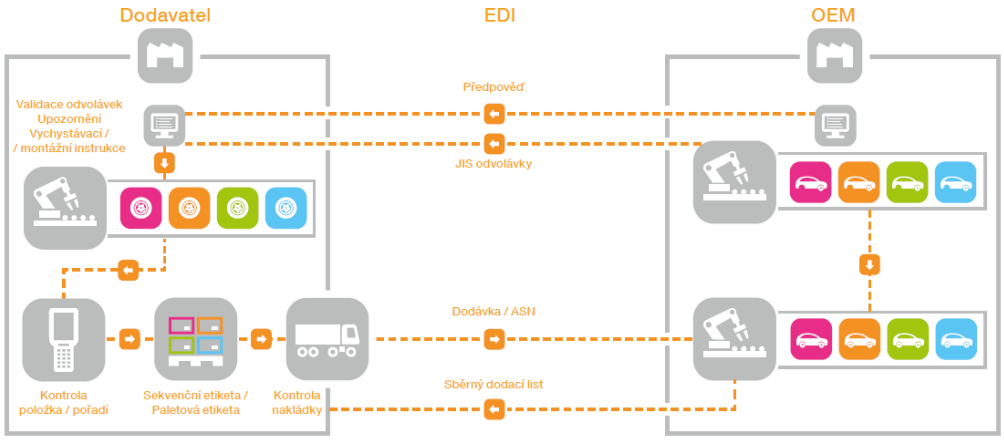


Obrázek 3.4: Ilustrační zobrazení fungování PORTALu [4]

Na obrázku 3.4 je zjednodušeně popsána funkčnost PORTALu. Pomocí internetu zde komunikují dodavatelé a zákazníci, kdy zákazník nejdříve provede objednávku materiálu v aplikaci DCIx. Ta následně upozorní dodavatele na objednávku a čeká se na potvrzení objednávky od dodavatele. Poté, co dodavatel dokončí přípravu materiálu, odešle zásilku ze skladu i s elektronickým dodacím listem. Ten poté slouží u zákazníka pro kontrolu a urychlení přijmu materiálu od dodavatele.

3.7 Řízení denních odvolávek a sekvenčních dodávek

Posledními oblastmi jsou moduly JIT (*Just In Time*) a JIS (*Just In Sequence*) určené pro automatické řízení dodávek s využitím stejnojmenných metod. Metoda JIS spočívá v tom, že odběratel je zásobován k jednotlivým výrobním linkám v přesně stanoveném čase, pořadí a množstvím, které je na finální výrobek v danou chvíli potřeba. JIT slouží k minimalizaci zásob na skladu odběratele, a proto si od dodavatele nechává dovézt přesně takové množství zásob, které na danou dobu potřebuje. Cílem těchto řešení je zajištění přesné expedice včetně označení výrobků a elektronických zpráv podle požadavků zákazníka. Největšími výhodami jsou přednastavené dodavatelské koncepty pro automobilky, odstranění manuální práce při přípravě a zpracování dodávek a nakonec stoprocentní přesnost expedice a eliminace záměn.

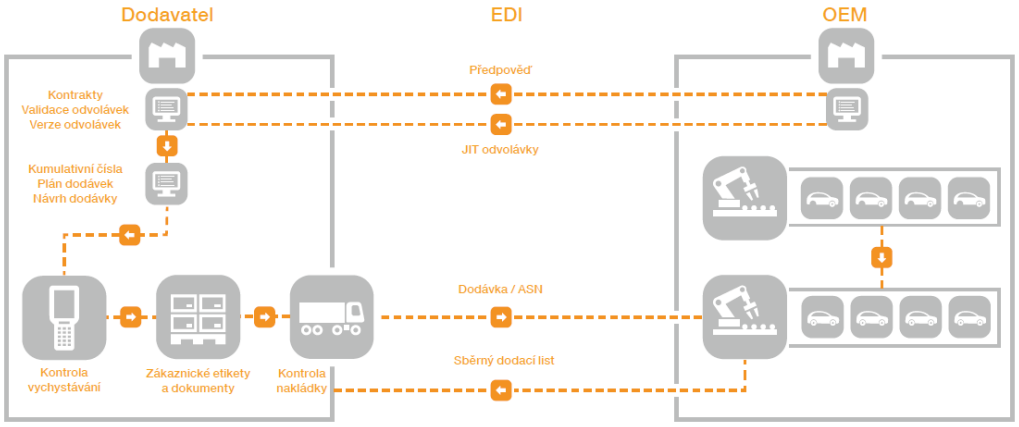


Obrázek 3.5: Ilustrační zobrazení fungování JIS [3]

Obrázek 3.5 znázorňuje výměnu dat mezi dodavatelem a odběratelem využitím metody JIS. Zde je odběratel označen pod zkratkou OEM (*Original Equipment Manufacturer*), která v tomto případě označuje některého z výrobců automobilových konstrukcí pro zcela nové auto. Na obrázku lze tedy vidět, že odběratel nejdříve oznámí plán výroby a poté jednotlivé linky žádají o dodání materiálu, jenž bude v blízké budoucnosti potřeba. U dodavatele v té chvíli začne ověřování odvolávek a začne se připravovat materiál či polovýrobek k expedici v požadovaném pořadí. Po správném seřazení jsou na zásilku přidány etikety a po konečné kontrole je zásilka odeslána. Jakmile odběratel obdrží zásilku, pošle zpětně potvrzený dodací list.

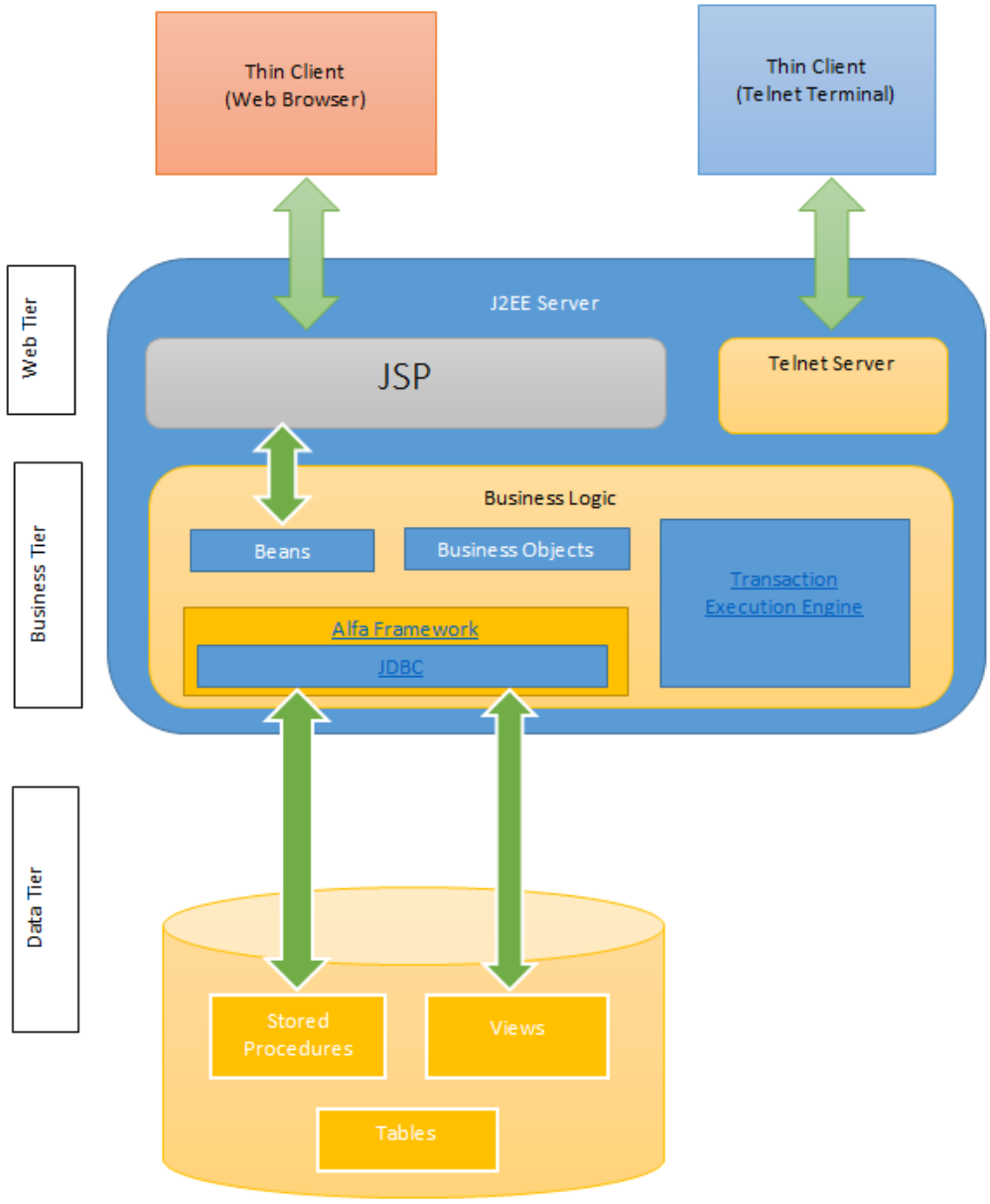
Funkčnost modulu JIT (viz Obrázek 3.6) je v mnoha ohledech podobná

jako předchozí metoda. Rozdílem je skutečnost, že odvolávky od zákazníka nechodí od jednotlivých linek, ale z centrálního řízení výroby. U dodavatele dochází ke kontrole odvolávek. Posléze se naplánuje a navrhne rozeslání dodávek. Odeslání zásilky probíhá téměř identicky pouze s rozdílem, že se posílají zákaznické etikety a dokumenty místo sekvenčních či paletových etiket.



Obrázek 3.6: Ilustrační zobrazení fungování JIT [3]

4 Architektura DCIx



Obrázek 4.1: Schéma architektury produktu DCIx [9]

Architektura produktu DCIx (viz Obrázek 4.1) se rozděluje na tři hlavní třídy. Tyto třídy si mezi sebou vzájemně vyměňují data. Nakonec všechna data musí být interpretována některým z klientů na uživatelské straně. Kla-

sickým klientem je webový prohlížeč libovolného druhu, i když nejlepší kompatibilitu má prohlížeč od Microsoftu – Internet Explorer nebo jeho novější verze s názvem Edge. Druhým velice používaným klientem je klasický telnet, určen převážně jen na provádění jednotlivých transakcí. Pracuje se ale i na mobilní aplikaci. Celá aplikace běží na serveru. Přesněji na webovém open source serveru nazývající se Apache Tomcat[1]. Tomcat je založen na jazyce Java, javových servletech, JSP(*Java Server Pages*) a EJB(*Enterprise JavaBeans*).

4.1 Datová třída

Datovou třídu tvoří databázová vrstva. Tato vrstva je vytvořena produktem SQL Server od Microsoftu, který používá rozšíření jazyka SQL (*Structured Query Language*) s názvem T-SQL (*Transact-SQL*). Nachází se zde dvě databáze. První databází je archivní a jak už název napovídá, tak slouží k uchovávání zálohy pro aktivní databázi. Tato databáze je nutná pro rychlou opravu v případě nenadálé chyby či ztráty dat v hlavní databázi. Dále slouží k analyzování množství dat a dopomáhá tak k optimalizaci ukládaných dat v databázi. Analýza nárůstu dat je důležitá především pro zákazníka, zejména pro výhled na budoucí investici pro zvětšení místa pro ukládání dat. Dá se proto včas řešit problém s docházejícím místem na discích. Druhá databáze je nazývána jako hlavní či aktivní databáze. V této databázi se nachází všechna data ukládaná aplikací. Mezi těmito daty se dají nalézt všechny proběhlé transakce, které mohou obsahovat změny ve skladu či příjem zboží. Dále jsou zde uložena všechna data o uživatelích, skladu, firmě, zboží a mnoho dalších. Dá se říct, že se v databázi nachází všechna potřebná data pro správný běh aplikace a k tomu, aby každá akce provedená v aplikaci byla snadno dohledatelná.

Datová třída komunikuje s další třídou pomocí zmíněných pohledů a uložených procedur, kdy dochází k dotazování pomocí vytvořeného Alfa Frameworku, který obsahuje komponenty pro bezpečné dotazování a ukládání dat do databáze.

V databázové vrstvě se nenachází pouze uložená data, ale je zde i mnoho uložených procedur, funkcí, pohledů a tzv. spouštěčů (*triggers*).

[1] `https://tomcat.apache.org/`

4.1.1 Uložené procedury

Procedury jsou v podstatě uložené části kódu, které můžeme volat kdykoliv je potřeba vykonat daný kód. Do procedury mohou vstupovat parametry, ovlivňující chování procedury. Procedury nemusejí mít žádnou návratovou hodnotu.

Uložené procedury jsou uloženy přímo v databázi spolu s ostatními objekty. To umožňuje jednodušší distribuci aplikace, ale přispívá to i ke spolehlivosti, neboť aplikační logika je zabezpečena a zálohována stejně spolehlivě jako samotné údaje. Uložené procedury se ukládají do databáze zpravidla v předkompilované podobě [10].

4.1.2 Funkce

Funkce je speciálním případem uložené procedury, která na rozdíl od procedury vrátí bloku, v němž byla zavolána, nějakou hodnotu. Tato hodnota je získána nebo vypočítána v těle procedury [10].

4.1.3 Pohledy

Pohled je možné definovat jako předpis pro vytvoření podmnožiny dat z jedné či více databázových tabulek. Využívají se například pro zpřístupnění jen určité podmnožiny údajů z databázové tabulky. Pohled může obsahovat i sjednocené údaje z více tabulek, takže uživatelé k nim mají přístup, aniž by museli používat komplikované složené příkazy jazyka SQL [10]. Obecně nelze přímo modifikovat data v pohledu.

4.1.4 Spoušť

Pojmem spoušť rozumíme uloženou proceduru, která se automaticky aktivuje v případě předem definované události, která nastává při manipulaci s údaji. Spouště se nikdy nepouštějí přímo. Spouště se mohou použít ke kontrole zadávaných údajů, pro zajištění datové integrity a podobně [10].

Událostí pro aktivování spouště může být například smazání nebo změna dat v řádku či v tabulce. Spoušť se může provést dvěma způsoby. Buď se spustí před provedením změny v tabulce, nebo se provede po změně dat.

4.1.5 Vysvětlení pojmu SQL a T-SQL

Jazyk SQL se využívá převážně v relačních databázích. Jedná se o strukturovaný dotazovací jazyk. Skládá se ze tří důležitých složek DML (*Data Ma-*

nipulation Language), DCL (*Data Control Language*) a posledním je DDL (*Data Definition Language*).

Složka DDL slouží k definici datových struktur. Pomocí této složky se v jazyce SQL dají vytvořit objekty, smazat tyto objekty, smazat obsah tabulek nebo přejmenovávat námi vytvořené tabulky. Při vytváření tabulky musí být specifikován název tabulky, jména sloupců a jejich datový typ.

Druhá složka DML napomáhá manipulovat s daty uložených v databázích. Dovoluje především čtyři nejdůležitější operace, kterými jsou smazání dat, upravení dat, vložení nových dat nebo jen jejich výběr z tabulky.

Poslední je DCL starající se o přiřazování přístupu uživatelů k datům. Díky tomu se mohou libovolně odebírat či přidělovat práva k oběma předchozím složkám. Lze odebrat práva například na mazání, vkládání či upravování dat v tabulkách.

Jazyk T-SQL od Microsoftu rozšiřuje standardy klasického SQL. Přivádí do něj definování lokálních proměnných, funkce podporující zpracovávání textu, vytváření procedur a přidává větvení v kódu pomocí klíčových slov **IF** a **ELSE**. Samozřejmě se nejedná o všechny změny, protože těch je daleko více.

4.1.6 Transakce, žurnály a návrat změn

Pokud se probírá relační databáze a jazyk SQL nesmí se zapomenout na důležitou součást, která k této problematice patří, a to jsou transakce. Transakce je skupina příkazů převádějící databázi z jednoho konzistentního stavu do druhého. Nemůže se tak stát, že by se provedla například jen polovina příkazů v transakci. Transakce musí splňovat atomičnost, konzistentnost, izolovanost a trvalost. Atomičnost pro transakci znamená, že nemůže být dále dělitelná. Konzistentní vlastnost přidává podmínku, že při provedení nebo po dokončení nesmí být porušeno žádné z integritních omezení. Předposlední podmínka pro izolovanost říká, že operace uvnitř transakce musí být ukryty před vnějšími operacemi. Poslední podmínkou je trvalost, ta dodává, že po úspěšném dokončení transakce musí provedené změny být uloženy v databázi a už nemohou být ztraceny. Průběh transakce a původní stav před provedení transakcí je zapisován do žurnálu. V případě chyby znemožňující doběhnutí transakce se provede vrácení všech změn (*Rollback*), pro opětovné nastolení konzistentního stavu databáze.

Typickým příkladem databázové transakce je převod finanční částky z jednoho účtu na druhý. Převádí-li klient A finanční částku na účet klienta B, musí se tato částka nejdříve odečíst z účtu klienta A, a poté přičíst na účet klienta B. V případě nenadálé chyby v průběhu transakce se může

stát, že se finanční částka odečte z účtu klienta A, ale kvůli nastalé chybě nelze tyto peníze uložit na účet klienta B [10]. Takový stav není přípustný, a proto se používá žurnál pro zjištění, zda transakce doběhla úspěšně a pokud nedoběhla úspěšně vrátí se všechny provedené změny zpět.

4.1.7 Relační databáze

Výše bylo zmíněno, že se tento jazyk nejvíce využívá v relačních databázích. Relační databáze jsou založené na relačním modelu, ze kterého vyplývají čtyři kardinality vztahu mezi jednotlivými tabulkami. Vztahy jsou tvořeny pomocí dvou typů klíčů, ale celkově existují typy tři. Prvním se nazývá primární klíč a jedná se o jednoznačný identifikátor záznamu v tabulce. Stejná hodnota či kombinace hodnot se proto nemůže vyskytovat u jiného záznamu ve stejné tabulce. Primární klíč může tvořit jeden sloupec či kombinace více sloupců. Druhým klíčem je cizí nebo nevlastní klíč. Ten vyjadřuje vztah mezi dvěma tabulkami tím způsobem, že hodnota ve vybraném sloupci, který tvoří cizí klíč, musí existovat v jiné tabulce jako hodnota primárního klíče. Posledním klíčem je kandidátní klíč, ten je úzce spjatý s primárním klíčem, protože z existujících kandidátních klíčů se vybírá nejlepší možný primární klíč pro danou tabulku. Nevybrané kandidátní klíče se poté mohou nazývat alternativní klíče. Vztahy jsou tedy určeny pouze pomocí primárního a cizího klíče.

Prvním vztahem, který může vzniknout, je vztah označován jako 1:1. Takový vztah říká, že jednomu záznamu v první tabulce odpovídá právě jeden záznam z druhé tabulky a to platí i naopak. Tento vztah vzniká propojením dvou primárních klíčů v tabulkách. V reálném světě to může být ukázáno na příkladu, kdy jeden člověk vlastní pouze jeden občanský průkaz a jeden občanský průkaz patří pouze jednomu člověku.

Propojením primárního klíče a cizího klíče nám může vzniknout relace 1:N. Pro toto spojení platí, že jednomu záznamu z tabulky A může odpovídat více záznamů z tabulky B. Naopak vztah platí následovně, pro jeden záznam z tabulky B existuje opět pouze jeden záznam v tabulce A. Toto je nejpoužívanější typ relace, protože odpovídá nejvíce situacím z reálného světa. Příklad takového vztahu v reálném světě je, že jeden časopis obsahuje několik článků a naopak jeden článek patří pouze k jednomu časopisu.

Předposlední vztah se popisuje jako M:N. Z tohoto spojení vyplývá, že pro jeden záznam z tabulky A smí odpovídat více záznamů z tabulky B a naopak, že pro jeden záznam z tabulky B může existovat více záznamů z tabulky A. Tento vztah se musí pro jeho implementaci rozložit na dva vztahy 1:N. Příklad takového vztahu v reálném světě může být například

student, který chodí na více předmětů a z druhé strany, že na jeden předmět chodí více studentů.

Posledním typem vztahu je neexistující spojení mezi tabulkami.

4.2 Byznys třída

Tato třída se stará o většinu funkcí produktu DCIx. Zajišťuje most mezi akcemi uživatele a daty u uživatele. V této třídě lze nalézt již zmíněný Alfa Framework, který umožňuje objektově-relační mapování. To zajišťuje automatickou konverzi data mezi relačními databázemi a objektově orientovaným programováním. Největší výhodou tohoto frameworku je to, že programátor může pouze volat jeho funkce. Alfa Framework posléze sám sestaví požadovaný dotaz automaticky a následně vrátí i jeho výsledek. Takto sestavený dotaz je zabezpečen proti útokům jako je například SQL Injection a navíc se předchází zbytečným chybám programátora. Samotný Framework nemá žádné omezení pro práci s daty v databázích, a proto je možné s ním provádět libovolné operace. Další důležitou částí je Transaction Execution Engine. Ten se zjednodušeně stará o zpracovávání transakcí a jejich úspěšné proběhnutí.

4.2.1 Transakce v DCIx

Transakce slouží k dynamickému definování některých částí aplikace. Tyto části zahrnují například přesun materiálu ve skladu nebo příjem a výdej jednotlivých položek. Transakce se skládají z jednotlivých modulů. Každá transakce má uložený záznam s definicí jejího složení v databázi v tabulce Transaction_Definition a pomocí tohoto záznamu je v aplikaci poté spouštěna. Každé spuštění transakce musí být také evidováno v databázi, aby se jednotlivé akce uvnitř skladu daly zpětně vystopovat. Pro tento účel slouží tabulka s historií transakcí s názvem Transactions. Jednotlivé transakce lze také snadno exportovat z aplikace a následně se mohou importovat do jiného projektu. Jelikož se jedná o důležitou funkci celé aplikace, není proto možné abychom spustili transakci přímo naostro s nejistou, zda jsme nastavili transakci správně. Při vytváření transakce tak můžeme spustit danou transakci v ladícím režimu, kdy vidíme všechny detaily, které se provádí. Pro programátora to má ještě větší výhodu, jelikož vytvoření nového modulu je nutno také otestovat a pomocí ladícího režimu lze hledat chybu rovnou i v kódu, protože zjištěný řádek s chybou se rovnou objeví v programovacím prostředí.

4.2.2 Final Values

Tyto hodnoty se používají v transakcích pro předávání používaných hodnot mezi jednotlivými moduly. Hodnoty jsou vyplňovány v modulech. Mohou být ovlivněny i uživatelským vstupem, kdy se například zadá číslo štítku zboží a v modulu se automaticky doplní název zboží i jeho pozice ve skladu.

4.2.3 Moduly

Moduly mají ve většině případů jen minimální funkčnost proto, aby se snadněji daly skládat do složitějších celků, které tvoří transakci. Pro každý modul je vytvořená speciální třída, tyto třídy jsou pojmenovávány shodně s názvem modulu. Každý modul navíc může obsahovat parametry. Tyto parametry se chovají pro každý modul jinak. Některé moduly dovolují zapisovat do argumentu hodnotu, u jiných se pouze zobrazí daná hodnota v argumentu a u některých se daný argument ignoruje.

4.3 Webová třída

Poslední třída je rozdělena na dvě hlavní komponenty. První komponenta je Telnet server, který se stará o komunikaci prostřednictvím telnetu. Jak už bylo výše zmíněno, pokud se aplikace používá přes telnet, zobrazí se uživateli jen dostupné transakce, které může spouštět. Transakce si volí tím, že píše identifikační čísla dané transakce. Pro přístup na telnet musí mít uživatel oprávnění. Druhou komponentou je JSP. Tato část se stará o definici vzhledu stránek, zobrazující se v prohlížeči u uživatele. Zároveň zachycuje všechny akce prováděné uživatelem a předává je dál do byznys třídy.

4.3.1 Telnet

Jedná se o protokol a stejnojmennou aplikaci. Protokol pracuje na aplikační vrstvě model TCP/IP a umožňuje připojení uživatele ke vzdálenému počítači pomocí textového uživatelského rozhraní. Pro komunikaci se nejčastěji používá port s číslem 23. Program realizuje komunikaci pomocí tohoto protokolu a samotný program se podobá terminálu.

4.3.2 JavaServer Pages

JSP je technologie sloužící k vytváření dynamických webových stránek. Technologie je založena na programovacím jazyce Java. Při vývoji se primárně využívá HTML, do kterého je vkládán kód napsaný v Javě.

5 Návrh logiky komunikace

Při návrhu výměny dat mezi aplikací DCIx a řídící jednotkou je nutné se nejprve zamyslet nad reálným používáním této funkcionality ve skladu a podle toho navrhnout optimální řešení. Musí se zohlednit jak se pracuje a bude pracovat s DCIx při komunikaci se zařízením a zároveň se nesmí zanedbat pohled na pracovní postupy při ovládání poloautomatického vozíku.

5.1 Použitelnost řídící jednotky ve skladu

Ve skladu se bude pohybovat jeden či více vozíků a v každém z nich bude přimontován KBOX. Na tato zařízení se musejí přenášet data o úkolech pro daný vozík nejlépe tak, aby každý operátor vozíku měl přesné a dostačující údaje o svém úkolu. Informace se zobrazují na displeji zařízení co nejpřehledněji. Větší přehlednosti lze dosáhnout pomocí barevného rozlišení textu zobrazovaného na displeji. Na displeji by tak měly být minimálně informace o souřadnicích ve skladu a názvu daného úkolu. Ve chvíli, kdy operátor vozíku má všechny potřebné informace o svém úkolu, začne jej plnit. V tento moment by měl skladový systém čekat, než danou úlohu provede. To vede k rozhodování o tom, jak poznat splnění cíle. Buď zde může být člověk, který bude hlídat dokončení úlohy a posléze to potvrdí skladovému systému, nebo po každém ukončení úlohy dojde potvrdit dokončení sám operátor vozíku. Obě tato řešení nejsou bohužel příliš optimální pro výkonnost skladu. Nabízí se však ještě řešení oznámení ukončení úkolu přes řídící jednotku. To lze zařídit stisknutím jednoho z tlačítek, která jsou umístěna na dotykovém displeji, jak se již na začátku bakalářské práce uvádělo. Toto řešení je daleko účinnější, jelikož nenastává žádná prodleva v komunikaci se skladovým systémem a je proto možné ihned pokračovat v práci dále.

5.2 Uvažování nad funkcemi DCIx

Předešlý pohled přibližuje možné využití komunikace s informačním systémem. Nyní je nutné vzít v potaz fungování aplikace DCIx a sloučit tyto dva pohledy v efektivně fungující celek.

Nejdříve je potřeba vyřešit způsob výměny dat mezi zařízením a aplikací. Z kapitoly o řídící jednotce vyplývá, že se musí použít TCP. Aplikace DCIx již komunikuje s některými zařízeními přes tento protokol, a proto je již plně

vyvinut a je připraven k použití. Bude ovšem potřebné udělat rozšíření pro podporu komunikace s tímto určitým zařízením. Způsob návrhu spojení je tímto ujasněn.

Aplikace DCIx zpracovává změny ve skladu pomocí již zmíněných transakcí. Tyto transakce jsou pro funkčnost komunikace klíčové, jelikož veškerá komunikace se zařízením bude probíhat jen ve vytvořených transakcích. Každá transakce se skládá z více modulů, kdy každý má jinou funkčnost. Vhodné by bylo, kdyby existoval modul, který by uměl komunikovat s řídící jednotkou. Poté by se dala tato funkčnost jednoduše rozšířit u všech potřebných transakcí tím, že by se do nich přidal jen tento modul. Jak již bylo zmíněno, tak způsob pro komunikaci pomocí TCP již v aplikaci existuje a není tedy náhodou, že pro tento úkol byl vytvořen univerzální modul. Za úvahu stojí, zda tento modul bude splňovat všechna kritéria potřebná pro efektivní komunikaci se zařízením či je potřeba vytvořit modul nový.

5.3 Uvažování nad modulem

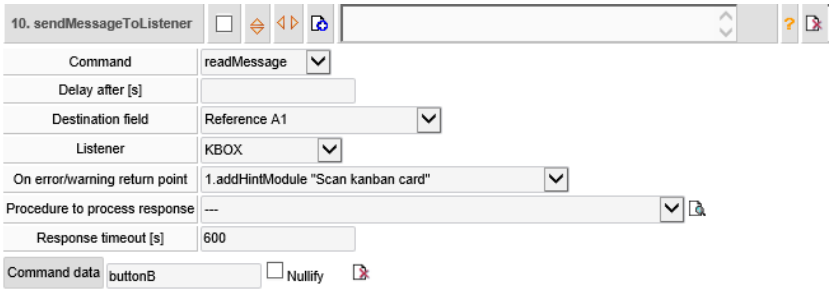
Od navrhovaného modulu se očekává, že bude umět posílat zprávy na zařízení. Dále by měl umožňovat výběr zprávy, která se má v daný čas odeslat. Nebylo by špatné, kdyby tento modul uměl i zprávy přijímat. Z úvahy nad použitelností řídící jednotky ve skladu vyplývá, že je nezbytné zařídit v modulu způsob, jak čekat než se zadaný úkol dokončí. Nesmí se zapomenout také na důležitou vlastnost tohoto modulu a tou je že musí umožňovat odesílat dynamická data a ne pouze statická. Dynamická data jsou taková data, která mohou měnit svůj obsah podle aktuálního stavu. Statická jsou naopak data, která se zadají při vytváření transakce a při běhu transakce už nemají možnost na změnu.

5.3.1 Modul sendMessageToListener

Existující modul má název sendMessageToListener (viz Obrázek 5.1) a umožňuje nastavit naprogramovanou funkci pro jednoho z připojených posluchačů (zařízení). Tyto funkce mohou zprávy přijímat i posílat. Modul dále podporuje výběr procedury, spouštějící se na začátku tohoto modulu, a zvolit dobu, po kterou má aplikace čekat na zprávu ze zařízení.

K tomuto modulu lze přidat i důležitý parametr se jménem Command Data, tvořený z textového pole. Data zadaná v tomto poli jsou předána dále do modulu a může s nimi v kódu být zacházeno podle potřeby. Podle zjištěných informací o modulu by měl potřebám komunikace se zařízením vystačit. Jediným problémem zůstává fakt, že do parametru Command Data

nelze vkládat data dynamicky uvnitř modulu. Tento problém však lze vyřešit pomocí procedury, která data parametru předá automaticky za běhu transakce ještě před spuštěním modulu sendMessageToListener. Tuto proceduru je možno spustit prostřednictvím jiného modulu s názvem executeProcedure. Tento modul je vytvořený přesně pro spouštění podobných procedur. Pro fungování a smysl procedur bude muset být tento modul před modulem pro odesílání dat. Odesílání dynamických dat se tak docílí dvěma moduly a pokud se bude posílat zpráva se statickým obsahem, stačí pouze jeden modul. Pro tuto chvíli není nutné dodělávat do aplikace nový modul, protože s tímto lze dosáhnout všech důležitých funkcí.



Obrázek 5.1: Čekání na stisknutí tlačítka v modulu sendMessageToListener

5.4 Kompatibilita souřadnic skladu

Předposlední věcí, kterou je nutné vyřešit, je kompatibilita popisování pozic balíčků ve skladu u řídící jednotky a v aplikaci. Řídící jednotka umožňuje inkrementální nebo segmentované počítání. V aplikaci se eviduje umístění ve skladu čtyřmi hodnotami – názvem skladu, sloupcem, řadou a podlažím. Pokud by se mělo použít segmentované počítání, je třeba znát jak pozice v řadě, tak umístění v rámci sloupce. Takovéto řešení není příliš vhodné z důvodu chybějící informace o pozici v řadě, která je v aplikaci dána kombinací tří zmíněných hodnot. Je daleko vhodnější využít inkrementační způsob. Odůvodněním je, že sice není vedena hodnota o pozici ve skladu, ale nemusí se uvádět hodnota o pozici v rámci sloupce. Z tohoto důvodu se může označení sloupce využít jako hodnota pozice v dané řadě. Splňuje se tím i podmínka pro inkrementační počítání, že každá pozice musí mít unikátní hodnotu v rámci své řady.

5.5 Použitelnost DCIx ve skladu

Když je vyřešená komunikace a způsob předávání dat přes transakce a moduly, zbývá zamyšlení nad tím, jak se bude s tímto řešením pracovat ve skladu u počítače s běžící aplikací. Do skladu například přijde nové zboží. Zboží se příjme na sklad a poté se spustí transakce pro zaskladnění. Skladník naskenuje první zboží, aplikace mu nabídne nejlepší umístění v rámci skladu a po jeho zvolení se odešle požadavek na KBOX o vyzvednutí položky a následné uložení balíčku. Bylo by dobré, kdyby ve skladu byl skladník připravující zboží k uložení a operátor vozíku se soustředil jen na rozvážení balíčků po skladu.

5.6 Shrnutí návrhu

Shrnutí požadavků na funkcionalitu a návrhu řešení je následující. Aplikační řešení komunikace přes TCP bude rozšířeno o podporu komunikace se zařízením. Musí umět přijímat a odesílat data s využitím modulu s názvem sendMessageToListener. Pro dynamičnost vstupu dat je potřeba aplikaci doplnit o procedury, které se postarají o dynamické zpracování a předání dat do daného modulu. Komunikace musí umět reagovat na stisknutí tlačítek na displeji řídící jednotky, takže i to musí být součástí rozšíření aplikace.

6 Implementace

Před začátkem implementace je potřeba určit postup, jakým se bude práce vyvíjet. První věcí, která se musí vytvořit je posluchač pro KBOX v aplikaci, aby se mohl zvolit ve zmíněném modulu. Následně je nutné v posluchači nadefinovat zpracovávání příchozích zpráv. Potom se vyvine vytvoření zpráv k odeslání na zařízení. Nakonec, když toto řešení bude hotové a funkční, vytvoří se procedury pro dynamické doplnění dat do Command Data.

6.1 Rozšíření databázových tabulek

Pro funkčnost posluchače se musí nejdříve vložit hodnoty do databáze. Odtud jsou hodnoty čtené v aplikaci a nebylo by proto možné bez těchto hodnot nalézt a používat nově vytvořeného posluchače.

Pro tento úkol jsou důležité čtyři tabulky (viz Obrázek 6.1). Nejdříve se vytvoří nový záznam v tabulce s dekodéry. Nový dekodér je pojmenován po zařízení KBoxDecoder. Tabulka slouží k nalezení javovské třídy pro dekódování zpráv.

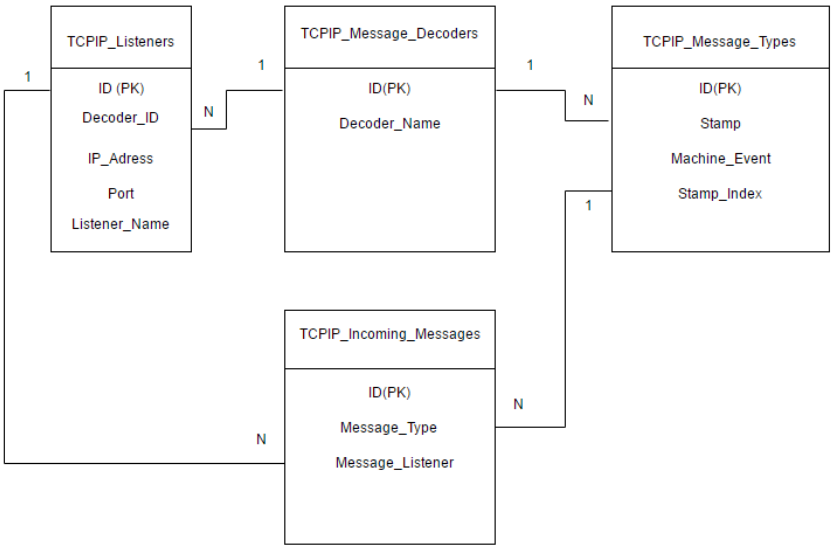
V tabulce s posluchači se musí vytvořit nový záznam. Záznam obsahuje především název posluchače, IP adresu, port pro komunikaci se zařízením a cizí klíč s identifikátorem dekodéru zpráv pro zařízení. Poslední tabulkou, kam se musí doplnit hodnoty, je TCPIP_Message_Types. Slouží k rozpoznání typů příchozích zpráv. Tabulku bylo nezbytné rozšířit o dva sloupce – Stamp a Stamp_Index. Důvodem k tomuto rozšíření bylo zjednodušení rozpoznávání zpráv využitím faktu, že každá zpráva má jedinečný identifikátor. Pro tento identifikátor je určen první zmíněný sloupec. Druhý sloupec označuje pozici tohoto identifikátoru ve zprávě. Použité identifikátory jsou téměř všechny jednopísmenné až na jednu výjimku. Výjimkou jsou příchozí zprávy po stisknutí tlačítek, kdy je nutno rozpoznat jak typ zprávy, tak i stisknuté tlačítko. Poslední hodnotou, kterou je nutno doplnit, je sloupec Machine_Event, obsahující pojmenování zprávy. Dá se říct, že tento sloupec je názvem typu příchozí zprávy.

Do poslední tabulky TCPIP_Incoming_Messages není nutno nic doplňovat. Záznamy se do této tabulky doplňují automaticky z aplikace, protože slouží k záznamu příchozích zpráv ze zařízení. Může tak sloužit ke kontrole či hledání chyby při zpracování zprávy.

Po doplnění hodnot je nutno vytvořit SQL skript doplňující tyto hodnoty do tabulek automaticky, pokud ještě v tabulce neexistují. Tím se zajistí

funkčnost i na jiných projektech bez nutnosti opisování hodnot do tabulek ručně.

Nakonec se změny ve sloupcích v tabulce TCPIP_Message_Types musí přidat i do mapovacích tříd v Javě. Tyto třídy musí znát přesnou podobu tabulek pro jejich snadnější používání v kódu.



Obrázek 6.1: ERA diagram tabulek souvisejících s TCP komunikací

6.2 Vytvoření KBOX posluchače

Po doplnění databáze lze začít s vývojem posluchače v aplikaci. Tento krok byl jeden z nejnáročnějších, protože se musela důsledně prozkoumat funkčnost dosavadní implementace TCP připojení. Z průzkumu vyplynulo, že se musí založit třídy zvlášť pro přijímání dat a odesílání dat. Pro příjímání dat se musí navíc vytvořit třída dědící od třídy IOHandlerAdapter. Třída bude zachytávat zprávy ze řídící jednotky a poté je bude předávat dále do třídy zpracovávající příchozí zprávy. Konečná struktura implementace posluchače je na obrázku 6.2. Také bylo potřeba založit třídu pro klienta starající se o nastavení připojení posluchače. Tento klient také obsahuje metody vracející vytvořené instance tříd KBoxCodecFactory a KBoxProtocolHandler.

Dále se musela vytvořit hodnota s posluchačem ve třídě ListenerType. U hodnoty se musela specifikovat třída s klientem, třída pro odesílání zpráv a nakonec počáteční a ukončovací znak pro správné přijetí zprávy. Pro vytvoření typu se používá výčet (*enumeration*), ve kterém je hodnota s názvem

Kbox (viz Ukázka kódu 6.1) nadefinována. Výčet je tvořen seznamem pojmenovaných konstant, které definují nový datový typ. Objekt výčtového typu může uchovávat pouze hodnoty uvedené v seznamu. Výčet tedy umožňuje přesně definovat nový datový typ, který má pevný počet platných hodnot[12].

```
public enum ListenerType {

  Kbox  (KBoxClient.class,
         KBoxCommand.class,
         "" + (char)0x02,
         "" + (char)0x03
        ) {
  },
  ;
}
```

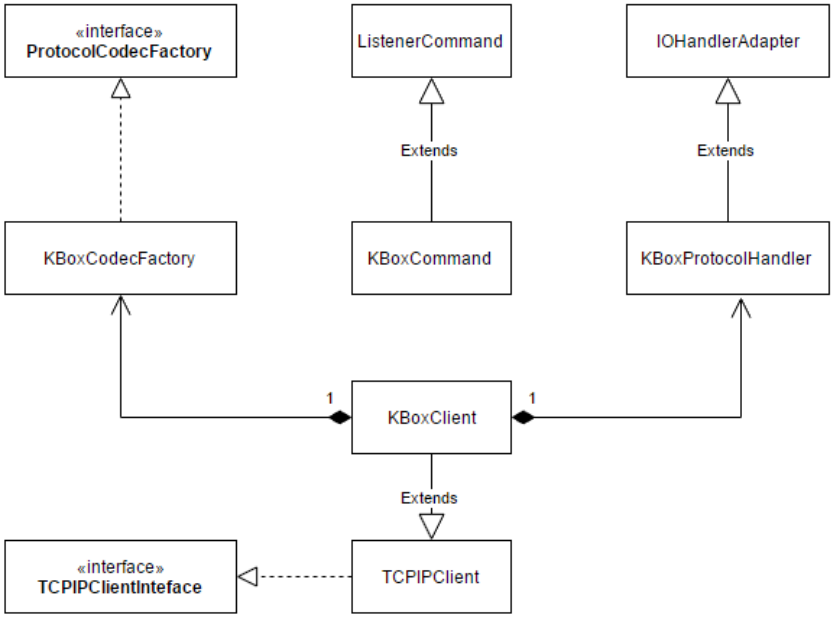
Ukázka kódu 6.1: Přidání hodnoty do výčtu

6.2.1 Přijímání zpráv

Důležitou součástí posluchače jsou metody pro přijímání zpráv. Tyto metody jsou obsaženy ve vnitřní třídě s názvem KBoxDecoder, nacházející se ve třídě KBoxCodecFactory. Třída obsahuje důležitou metodu addMessage, volanou při příchodu zprávy z daného posluchače. Metoda nejdříve oddělí z příchozí zprávy její identifikátor a zároveň zjistí pozici daného identifikátoru. Podle tohoto identifikátoru se posléze vyhledává typ zprávy v tabulce s typy zpráv.

Pro toto vyhledávání bylo potřeba dopsat metody do třídy MessageDecoderCache (viz Ukázka kódu 6.2) pro vyhledávání typů zpráv podle identifikátoru, pozice identifikátoru a typu použitého dekodéru. Návratová hodnota metody obsahuje všechny hodnoty sloupců vyhledaného záznamu. Především lze dohledat hodnotu ze sloupce Machine_Event podle, které lze zprávu zpracovat. Při spuštění služby TCPIP, zařizující komunikaci se zařízeními, se zavolá metoda pro vytvoření hashmapy obsahující typy zpráv. Posléze se během zpracovávání zprávy zavolá statická funkce pro dekódování, která prohledává vytvořenou hashmapu. Index v hasmapě tvoří již zmíněné informace – identifikátor, jeho pozice a používaný dekodér. Tyto informace jsou spojené podtržítkem do jednoho řetězce.

Nakonec se podle získaného typu zpráv naplní proměnná, která slouží k předání informací ze zprávy do modulů. Pro určení této proměnné existuje v modulu sendMessageToListener volba umožňující vybrat pole, ze kterého budeme moci přistupovat k získaným datům. Při příchodu zprávy po stisku



Obrázek 6.2: Zobrazení UML diagramu tříd potřebných ke KBOXu

tlačítka tato proměnná bude obsahovat jeho název. Při ostatních zprávách se zde bude nalézat obsah příchozí zprávy bez délky a identfikátoru.

```
public class MessageDecoderCache extends CacheSupport {
   public synchronized static MessageType decodeMessageTypeByStamp
       (Integer pDecoderId, String pStamp, int pStampPosition);

   protected Map<String, MessageType> createTypeDecodingStampIndex
       (MessageDecoderCollection pMessageDecoders);
}
```

Ukázka kódu 6.2: Doplněná metoda v MessageDecoderCache

6.2.2 Odesílání zpráv

Odesílání zpráv je obsaženo ve třídě KBoxCommand (viz Ukázka kódu 6.3). Prvními dvěma výrazy se vytváří instance tříd pro budoucí reference na objekty sloužící pro vyslání příkazu na zařízení. Každý příkaz, posílaný na zařízení, má vlastní vnitřní třídu. O použité třídě rozhoduje zvolený příkaz k vykonání v modulu sendMessageListener. Tyto vnitřní třídy se dělí na dva typy. Do prvního typu mohou vstupovat data pomocí zmíněného parametru, a proto tato třída musí dědit od třídy TCPIPParametrizedCommand.

Zde se data předají pomocí konstruktoru, kde jsou vyjádřeny jako listové pole řetězců. Pro toto předání je zde metoda getCommand, která vytváří instanci třídy příkazu a předá zde tyto data. Data v konstruktoru naplní globální proměnou v rodičovské třídě. Tím se umožní použití dat v jakékoli metodě této třídy. Druhý typ žádná vstupní data nepotřebuje. Jedná se například o příkaz ke smazání právě prováděných akcí. Tento typ tříd implementuje třídu TCPIPCommandOut, která neumožňuje získávat tyto data. Tyto třídy obsahují funkcionalitu jen pro zasílání neměnných zpráv. K poslání zprávy u obou typů slouží metoda buildMessage, která vrací finální podobu požadované zprávy před odesláním.

```
public class KBoxCommand extends ListenerCommand {
   public static final KBoxCommand clearDisplay = new KBoxCommand (
      new ClearDisplayCommand()
   );

   public static final KBoxCommand putaway = new KBoxCommand () {
      @Override
      public TCPIPCommandOut getCommand(List<String> pData);
   };

   private static class ClearDisplayCommand implements TCPIPCommandOut {
      @Override
        public String buildMessage();
      @Override
        public String getLoggingText();
   }

   private static class PutawayCommand extends TCPIPParametrizedCommand {
      public PutawayCommand (List<String> pData);
      @Override
        public String buildMessage();
      @Override
        public String getLoggingText();
   }
}
```

Ukázka kódu 6.3: Náhled na kód pro odesílání příkazů

6.3 Vytvoření procedur

Posledním krokem při vývoji bylo vytvoření Java procedur. První vytvářená procedura má za úkol předat správná data pro vyložení balíčku. Pro jednoduché vytvoření procedury se může oddědit vzor pro procedury s názvem

ProcedureBody. Poté stačí pouze přepsat dvě metody z rodičovské třídy. První metodou je setScriptedContext, kde se vytvoří instance procedury a poté se zavolá druhá přepisovaná metoda s názvem transactionXP. Uvnitř této metody se již nachází funkční kód, který se má postarat o veškerou funkcionalitu. Získávají se zde hodnoty položek toWarehouse a toPosition z *Final Values*. Po získání hodnot se volá metoda starající se o formátování těchto hodnot a vrací hotový řetězec, který se vloží do hodnoty commandData. Jednotlivé hodnoty jsou v tomto řetězci oddělené čárkou.

Druhá procedura slouží k předávání správných dat pro naložení balíčku. Tato třída dědí předešlou proceduru pro využití metody na formátování dat. Jinak funguje téměř identicky jako první procedura, jen bere hodnoty z položek fromWarehouse a fromPosition.

6.4 Problémy během vývoje

Během vývoje nastaly některé větší problémy, které oddálily dokončení práce. Prvním problémem bylo, když se v půlce vývoje přestaly spouštět testy v daném projektu. Nebylo tak možné usoudit, zda vývoj nerozbijí některé funkce aplikace a zda napsané testy fungují. Došlo se tedy k závěru, že bude rychlejší přenést celý vývoj na novou fungující verzi aplikace, než hledat příčinu chyby. Kvůli přechodu na novější verzi se musela i trochu upravit již dokončená implementace pro docílení kompatibility s novějšími třídami.

Další problém nastal, když se zkratoval napájecí zdroj k zařízení. Zaviněním mohlo být stáří použitého zdroje. Zařízení však nešlo vůbec spustit a musel se objednávat speciálně nový zdroj pro napájení. Po připojení k novému zdroji, musel být KBOX několikrát restartován než konečně naběhl a mohlo se pokračovat v práci.

7 Testování

Pro testování funkcionality řešení se použily jednotkové přesněji JUnit testy a automatické testy pomocí Selenium[1] nástroje. Testováním se navíc bude předcházet rozbití funkcionality během budoucího vývoje. Mimo testů bylo potřeba navrhnout jednoduché transakce, na kterých se mohla otestovat veškerá funkčnost.

7.1 Testovací transakce

Tyto transakce mohou mimo testování sloužit i jako názorná ukázka pro budoucí využití funkcionality v jiných transakcích. Vznikly transakce, které ověřují odeslání příkazu pro vyzvednutí balíčku, odeslání příkazu pro uložení balíčku, pro vypsání textu na displej, pro přijmutí skenovaných čárových kódů a reakci na stisknutí tlačítek. Nakonec vznikly dvě větší transakce ukazující použití v provozu. První slouží k přesunu balíčku po skladu a druhá předvádí vyskladnění balíčku. Druhá zmíněná transakce je použita pro automatické testování pomocí Selenia.

7.2 Jednotkové testy

Jednotkové testy jsou určené k testování menších částí zdrojového kódu. K tomuto testování se použil Framework JUnit[2], který se zaměřuje na vytvoření těchto testů v Javě. Napsané testy se starají o správnou funkčnost při získávání informací z příchozích zpráv, správné rozdělení typu zpráv a správné odesílání zpráv.

7.2.1 Testování příchozích zpráv

Tyto testy se zaměřují na metody ve třídě KBoxCodecFactory. Kontrolují správné výstupy při různých vstupech do metody pro získávání informací. Testy zkouší všechny korektní vstupy a následně i pár chybových, u kterých je minimální možnost výskytu. Takovým testem je například testování při vstupu prázdné zprávy, kdy se vrátí opět jen prázdná zpráva.

[1] `http://www.seleniumhq.org/`
[2] `http://www.junit.org`

7.2.2 Testování získání správného typu zprávy

Tento test se zaměřuje na metodu pro získávání typu zprávy z databáze podle vstupních argumentů. Pro tento test bylo potřeba v testu vytvořit falešnou třídu oddědenou z pravé třídy obsahující testovanou metodu. Důvodem bylo odstranění závislosti testované třídy na připojení k databázi. Místo toho se třída naplní statickými daty se záznamy z databáze. Poté už jsou napsané jen testy, které zkouší správné vstupy i vstupy, kdy žádný typ není nalezen.

7.2.3 Testování odchozích zpráv

Poslední jednotkové testování se zaměřuje na metody v třídě KBoxCommand. Testy zkouší všechny metody důležité pro odeslání zprávy. Kontrolují se zde tak metody pro odeslání příkazu k vyzvednutí balíčku, pro vypsání řádku na displej nebo pro smazání probíhajících příkazů. Opět se testují jen výstupy funkcí při různých vstupech.

7.3 Testování pomocí Selenium

Testy vytvořené pomocí tohoto nástroje slouží k automatickému testování webových aplikací. V DCIx tento nástroj využívá internetový prohlížet Google Chrome, nad kterým převezme kontrolu a simuluje naprogramované chování uživatele v aplikaci. Tento nástroj byl využit pro jeden test, který testuje klíčovou funkčnost v transakci.

Test byl navržen na otestování komunikace s KBOXem za běhu transakce pro vyskladnění. U této transakce se nejdříve vybere druh zboží, posléze se naskenuje číslo štítku. Z těchto informací se dohledá umístění balíčku a pomocí procedury zpracovávající tyto informace se data vloží do modulu pro odeslání příkazu. Následně se čeká na stisknutí tlačítka B pro potvrzení. Transakce pokračuje zadáním místa pro vyskladňované zboží. Opět se spustí procedura formátující informace o pozici ve skladu a tato data se předají do modulu pro odeslání zprávy. Odešle se příkaz k vyložení zásilky a znovu je prodleva než operátor potvrdí dokončení úkolu. Transakce se dokončí tím, že se balíček nastaví jako uzavřený a tím je balíček vyskladněn. Poté se může pokračovat zadáním nové zásilky nebo se transakce ukončí. V testu transakce končí.

V testu bylo potřeba využít simulovaného serveru, který se chová jako KBOX. Při této simulaci se mohou kontrolovat zprávy přicházející z transakce. Dále je možno nadefinovat odpověď serveru, takže se dá zkontrolovat i reakce na stisknutí tlačítka.

8 Dokumentace

Konečným bodem zadání bylo vytvoření dokumentace. Dokumentace je tvořená Javadoc komentáři u jednotlivých metod a tříd. Tato dokumentace je určena pro programátory, kteří budou rozšiřovat či upravovat napsaný kód. Je psána anglickým jazykem.

Druhá vytvořená dokumentace je určená pro koncové uživatele. Tato dokumentace je přístupná přímo z aplikace buď kliknutím na nápovědu k celé aplikaci, nebo kliknutím na nápovědu modulu, která otevře dokumentaci přímo k danému modulu. Dokumentace vždy otevře nové okno prohlížeče, nepřijdeme tak o aktuální stránku aplikace, na které se nacházíme. Zdokumentována byla nová funkčnost rozšiřovaného modulu sendMessageToListener. To zahrnuje popsání všech použitelných funkcí a popsání zpráv, které lze zadávat do parametru Command Data.

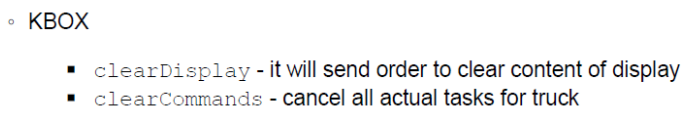
Uživatelská dokumentace je psána v souboru XML se strukturou připomínající HTML značky (viz Ukázka kódu 8.1). Na ukázce lze vidět seznam s jednou položkou. Tato položka má název KBOX a poté obsahuje vnořený další seznam s položkami. Další položky již informují o použitelných příkazech, které lze na zařízení poslat. Ukázka tohoto kódu z již vygenererované dokumentace je na obrázku 8.1. Za zmínku stojí i značka s názvem *code*, která změní styl písma tak, aby se poznalo, že jde o důležitou informaci. Napsaná byla verze v češtině a v angličtině. Každý jazyk má svůj vlastní adresář a soubor. V části dokumentace popisující komunikaci s řídící jednotkou byly nejvíce použity již zmíněné značky pro nečíslovaný seznam a odstavce. XML soubor je posléze převeden na HTML soubor, pro snadné zobrazování u klienta v prohlížeči.

Dokumentace modulů má vždy několik částí. Nejdříve dokument začíná názvem modulu a stručným popisem jeho funkčnosti. Následuje specifikace modulu. Zde jsou uvedeny nejdříve atributy a parametry, které jsou nastavené již automaticky. Poté následuje popis nastavení, která se mohou provést a na co mají tyto nastavení vliv. Popsány jsou v dokumentaci i vstupy a výstupy. Do vstupů se uvádí *Final Values*, s kterými se v modulu pracuje. Tyto paramtery mohou být povinné či nepovinné. Pokud je parametr povinný, ale nevstoupí do modulu, tak modul skončí na chybě. Tato chyba u nepovinného parametru nehrozí. Ve vstupech i výstupech lze nálezt i *Hidden Values*, které jsou pro uživatele transakce naprosto neviditelné. Nemůže s nimi tudíž manipulovat. Tyto hodnoty slouží pouze pro předávání informací mezi moduly. Nakonec jsou v dokumentaci uvedeny chybové stavy modulu a varovná

hlášení. U chybového stavu a varovného hlášení je vždy uveden kód chyby či varování a následně jejích vysvětlení.

```
<itemizedlist>
   <listitem>
      <simpara>KBOX</simpara>
      <itemizedlist>
         <listitem>
            <simpara> <code>clearDisplay</code> - it will send order to
                clear content of display</simpara>
         </listitem>
         <listitem>
            <simpara><code>clearCommands</code> - cancel all actual tasks
                for truck</simpara>
         </listitem>
      </itemizedlist>
   </listitem>
</itemizedlist>
```

Ukázka kódu 8.1: Zkrácený kód dokumentace v XML souboru



Obrázek 8.1: Ukázka vygenerované části dokumentace

9 Závěr

Dokončením práce se dosáhlo možnosti komunikace s řídící jednotkou. Uživatel aplikace nyní může snadno přijímat i odesílat zprávy na zařízení v jakékoli transakci použitím maximálně dvou modulů. Docílilo se tak naprosté kompatibility se všemi existujícími transakcemi pracujícími se skladem. Do transakcí stačí doplnit jen tyto dva moduly na místo pro vyložení či naložení balíčku a transakce je upravena. Aplikace umožňuje tedy automaticky doplnit informace o pozici balíčku ve skladu. Následně odeslat příkaz pro vyzvednutí balíčku na zařízení a posléze může aplikace čekat na potvrzení z vozíku, že byl balíček vyzvednut. Na displeji lze během této akce zobrazit informace o pozici balíčku a další informace.

Následné testování prováděné pomocí JUnit testů doplněných o test vytvořený v nástroji Selenium navíc zajišťuje, že vyvinutá funkcionalita bude nadále pracovat správně i při provádění rozšíření stávající funkčnosti.

Dokumentace poté pokrývá jak samotný vývoj, tak i způsob používání pro koncové uživatele dohledatelný v aplikaci. To dopomůže k rychlejšímu vývoji i k lepší použitelnosti u uživatelů.

V budoucnu se na tomto projektu mohou dělat různé úpravy či rozšíření v závislosti na rozdílných zákaznických požadavcích. Do té doby je docílená funkčnost dostačující.

Literatura

[1] The Addition of Explicit Congestion Notification (ECN) to IP. sep 2001. URL `https://tools.ietf.org/html/rfc3168`

[2] AIMTEC A. S., Hálkova 1185/24, 301 00 Plzeň: *Firemní dokumentace - DCIx*. 2014.

[3] AIMTEC A. S., Hálkova 1185/24, 301 00 Plzeň: *Firemní dokumentace - DCIx JIT/JIS*. 2014.

[4] AIMTEC A. S., Hálkova 1185/24, 301 00 Plzeň: *Firemní dokumentace - DCIx Portal*. 2014.

[5] AIMTEC A. S., Hálkova 1185/24, 301 00 Plzeň: *Firemní dokumentace - DCIx QMS*. 2014.

[6] AIMTEC A. S., Hálkova 1185/24, 301 00 Plzeň: *Firemní dokumentace - DCIx WMS*. 2014.

[7] AIMTEC A. S., Hálkova 1185/24, 301 00 Plzeň: *Firemní dokumentace - DCIx YMS*. 2014.

[8] Dordal, P. L.: 2014.
URL `http://intronetworks.cs.luc.edu/1/html/tcp.html`

[9] Kvapil, J.: *Programmer's manual*. AIMTEC A. S., Hálkova 1185/24, 301 00 Plzeň.

[10] Lacko, L.: *1001 Tipů a triků pro SQL*. Computer press a. s., 2011, ISBN 978-80-251-3010-0, strany 95, 209, 257, 260 a 262.

[11] Osterloh, H.: *TCP/IP Kompletní průvodce*. SoftPress, 2003, ISBN 80-86497-34-8, strana 232.

[12] Schildt, H.: *Java 8*. Computer press a. s., 2016, ISBN 978-80-251-4665-1, strana 417.

[13] STILL s. r. o., Štěrboholská 102, 102 19 Praha 10 – Hostivař: *Firemní dokumentace - Popis jednotky KBOX*.

[14] STILL s. r. o., Štěrboholská 102, 102 19 Praha 10 – Hostivař: *Firemní dokumentace - Telegram Exchange VNA Navigation System*. 2014.