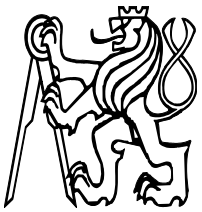


**ČVUT v Praze**

Fakulta strojní

Ústav přístrojové a řídicí techniky
Odbor automatického řízení a inženýrské informatiky

# Plně automatická linka pro produkci malonákladových knih

Diplomová práce
2003/2004

Vypracoval: Michal Málek

Vedoucí diplomové práce: Ing. Marie Martinásková, PhD.

Prohlašuji, že předkládanou diplomovou práci jsem vypracoval samostatně s tím, že její výsledky mohou být dále využity podle uvážení vedoucího diplomové práce jako jejího spoluautora. Souhlasím také s případnou publikací výsledků diplomové práce nebo její podstatné části, pokud budu uveden jako její spoluautor.

Datum:

.....
podpis

***Poděkování:***

Chtěl bych tímto poděkovat vedoucí své diplomové práce Ing. Marii Martináskové PhD. za cenné připomínky a rady při vypracovávání práce.

Dále bych chtěl poděkovat pracovníkům firmy Short Run Engineering GmbH za konzultace ohledně průběhu technologického procesu výroby knihy nejen z hlediska automatického řízení.

## Stručný souhrn obsahu práce

V rámci této diplomové práce je řešena problematika návrhu a realizace plně automatické linky pro produkci malonákladových knih a to především z pohledu řídicího systému celé linky a s tím související komunikace s podřízenými programovatelnými automaty jednotlivých strojů výrobní linky.

Koncept řešení uvedený v této práci je v mnoha ohledech nový a inovativní. Práce popisuje řešení pro budoucí výrobní systém s novou organizací výrobního procesu typu „Publishing on Demand“ (PoD) – publikování na zakázku.

Výsledkem této práce je funkční řešení prototypu řídicího systému výrobní linky. Pro usnadnění specifikace produktu, založené na formátu Job Definition Format (JDF), je k dispozici vytvořená webová aplikace v jazyce PHP. Řídicí systém, realizovaný pomocí softwarového produktu Genesis32 (GraphWorX) a VBA, nabízí kromě vlastního řízení procesu výroby knihy (na základě JDF specifikace) rovněž sofistikovaný systém pro správu zakázek, který je rovněž napojen na webovou aplikaci. Komunikace mezi řídicím systémem a programovatelnými automaty jednotlivých strojů probíhá pomocí B&R PVI OPC Serveru. Program jednotlivých automatů pro komunikaci s řídicím systémem je napsán v jazyce B&R Automation Basic.

Vytvořená řešení představují základ nově vyvíjeného nadřazeného řídicího systému výrobní linky pro produkci malonákladových knih ve firmě Short Run Engineering GmbH.

***Obsah:***

Seznam obrázků:

Seznam tabulek:

# 1 Úvod

Požadavek výroby knih o malém a nejmenším nákladu, a to řádově až jednotek kusů, představuje stěžejní problém realizace plně automatické linky. Takovéto linky, umožňující ekonomickou produkci knih o nejmenším nákladu, nejsou doposud běžně vyráběny.

Precizní konstrukční řešení z hlediska mechanických a technologických vlastností je nutnou podmínkou spolehlivé funkce celého systému, ale vlastní konstrukce jednotlivých strojů není předmětem této práce. Na druhou stranu je bezpodmínečně nutné znát technologický proces výroby knihy, a proto je v této práci alespoň přehledově uveden.

Přestože počet jednotlivých knih v rámci jedné zakázky (nákladu) je malý, počet různých zakázek určených k výrobě na dané lince je naopak relativně velký. Kromě nutnosti vytvoření systému zpracování/sledování jednotlivých zakázek, který je rovněž součástí této práce, s tím souvisí i zvýšené nároky na způsob zadávání jednotlivých parametrů technologického procesu výroby knihy.

Uvedené řešení je z pohledu specifikace parametrů založeno na standardu polygrafického průmyslu nazývaného Job Definition Format (JDF). Formát JDF je velmi rozšířen v oboru tisku, jeho aplikace v dokončovacích operacích zatím není běžná a zde navržená aplikace je tedy nová.

Z časového hlediska je pak rozhodující rychlost zadávání zakázek. Především s výhledem do budoucna je tedy vhodná existence určitého elektronického rozhraní mezi zadavatelem zakázky a průmyslovou knihárnou, tedy výrobní linkou, které by umožnilo vytvoření elektronické specifikace knihy, založené na JDF. Tyto požadavky jsou v této práci rovněž vyřešeny – ve formě webové aplikace, umožňující specifikaci zakázky (knihy) a její následné odeslání k výrobě.

Tato práce popisuje především softwarovou a hardwarovou realizaci řídicího systému. Dále je uvedeno řešení programového modulu určeného pro programovatelné automaty, který umožňuje komunikaci s nadřazeným řídicím systémem.

Všechna v této práci uvedená řešení jsem vytvořil v průběhu více než dvouleté spolupráce s firmou Short Run Engineering GmbH, (dále označovanou „firma SRE“) sídlící v Německu a patřící do skupiny Short Run Solutions. Firma SRE vyvíjí a vyrábí jednotlivé stroje zmiňované výrobní linky jak z hlediska konstrukce tak i automatizace. Řešení vytvořená v rámci této práce představují základ pro nově vyvíjený nadřazený řídicí systém výrobní linky.

Cílem kapitoly 2 je uvést obecný úvod do problematiky průmyslové výroby knih se zaměřením na dokončovací operace. V této souvislosti jsou také popsány koncepční změny při přechodu od systémů pro velké náklady na in-line systémy pro malé náklady a problematika komunikace se zaměřením na průmyslové komunikační systémy s detailnějším popisem vybraných komunikačních sběrnic.

Kapitola 3 se zabývá návrhem hardwarového a softwarového řešení pro dispečerskou úroveň řízení daného výrobního systému. Nejdříve jsou definovány požadované vlastnosti systému a poté je uveden popis navržených prostředků pro jejich splnění – webové aplikace umožňující zadávání dat, komunikační sběrnice, systému pro identifikaci produktu, řídicího systému, a programovatelných automatů.

Vlastní softwarové řešení dispečerské úrovně řídicího systému je komplexní a provázané s komunikací s podřízenými systémy a je proto uvedeno v samostatné kapitole 4.

Zhodnocení výsledků a přínosů této práce je uvedeno v kapitole 5. Pro úplnost je pak uveden seznam použité literatury v kapitole 6 a kapitola 7 obsahuje přílohy s doplňujícími informacemi pro uvedená řešení.

# 2 Obecný úvod do problematiky

## *2.1 Použité zkratky a označení*

| | |
|---|---|
| ADO | ActiveX Data Objects |
| B&R | Bernecker&Rainer |
| CAN | Controller Area Network |
| COM | Component Object Model |
| CPU | Computer Processor Unit |
| CRC | Cyclic Redundancy Check |
| CSMA | Carrier Sense Multiple Access |
| CSMA/CD | CSMA/Collision Detection |
| CSMA/CD+AMP | CSMA/Collision Detection and Arbitration on Message Priority |
| CSV | Comma Separated Values |
| FAN | Field Area Network |
| HMI | Human Machine Interface |
| I/O | Input/Output - vstup/výstup |
| IPC | Industrial Personal Computer - průmyslový počítač |
| JDF | Job Definition Format |
| LLC | Logical Link Control |
| MAC | Media Access Control |
| MES | Manufacturing Execution System |
| MIS | Management Information Services |
| Obr. | Obrázek |
| ODBC | Open Database Concept |
| OLE | Object Linking and Embedding |
| OPC | OLE for Process Control |
| OPLC | Operator PLC |
| OSI | OSI (Reference Model for Open System Interconnection) |
| PA | Programovatelný automat |
| PDC | Producer-Distributor-Consumer |
| PLC | Programmable Logical Controller = Programovatelný automat (PA) |
| PoD | Publishing on Demand – publikování na zakázku |
| PVI | Process Visualization Interface |
| RIO | Remote Input Output |
| SCADA | Supervisory Control And Data Acquisition |
| SCNM | Slot Communication Network Management |
| SQL | Structured Query Language |
| STP | Shielded Twisted Pair |
| SRE | Short Run Engineering GmbH |
| Tab. | Tabulka |
| TCP/IP | Transmission Control Protocol/Internet Protocol |
| TP | Twisted Pair |
| UDP | User Datagram Protocol |
| UTP | Unshielded Twisted Pair |
| VBA | Visual Basic for Applications |

*Poznámka: Vzhledem k charakteru této práce není důsledně dodržováno označení chráněných známek a práv copyright.*

## 2.2 Knihy o malém nákladu – Publishing on Demand (úvod do problematiky malých nákladů v kontextu hospodářské situace)

Vývoj v oboru tradičního tisku a především v průmyslové vazbě knih je v posledních letech charakterizován následujícími tendencemi:

- průměrné náklady knih stále klesají,
- zatímco současně roste počet vydávaných titulů

Podíl malých nákladů tedy roste a téměř 70% vydání má počet výtisků v rozmezí 500 – 5 000. Kvůli nízké efektivitě současných výrobních systémů při menších nákladech vzniká cenový tlak, který vede v konečném důsledku k razantnímu snížení možného zisku přepočteného na jednu knihy pro samotného výrobce. Po odečtení nákladů na reklamu a režii knihkupectví totiž zbývá pro tiskárnu zlomek (řádově jednotky až desetiny procent) prodejní ceny knihy.

Navzdory nástupu elektronických médií ale zůstává tištěné médium stále žádané. Přestože počty výtisků jednotlivých titulů klesají, nároky na kvalitu knih zůstávají vysoké.

Tato situace vyžaduje nové nároky na výrobu tištěných médií, jako například zkrácení doby výroby, minimální zásoby a dobu skladování, snížení ceny knihy a v neposlední řadě využití již existujících dat s možností aktualizace a personalizace obsažených informací.

Ekonomická výroba malých a nejmenších nákladů si tak vynucuje nasazení sofistikovaných automatizovaných řešení, sjednocení tisku a dokončovacích operací stejně jako novou organizaci procesu výroby knihy – jako například „Publishing on Demand“ (PoD) – publikování na zakázku.

PoD zahrnuje výrobu tištěného produktu od vytvoření dat (obsahu) přes vytištění (Printing on Demand – vytištění na zakázku) až po dokončovací operace (Binding on Demand nebo Finishing on Demand) při použití digitálních dat a digitálního tisku.

Pro umožnění tohoto způsobu výroby tištěných produktů (v daném případě knih) je tak třeba výrazně modifikovat stávající výrobní linky či vůbec vyvinout linky nové, koncepčně jinak řešené a uzpůsobené pro PoD.

## 2.3 Flexibilní systémy v dokončovacích operacích

Mechanické a logistické spojení jednotlivých strojů vykonávajících dílčí procesy při výrobě knih a v polygrafickém průmyslu vůbec je nasazováno již dlouhou dobu. Přesto není běžná realizace a nasazení takových systémů, kde jednotlivé stroje jsou nejenom mechanicky a logisticky spojeny ve výrobní lince, ale jednotlivá zařízení jsou schopna mezi sebou komunikovat (sdílet informace o aktuálním stavu jednotlivých parametrů) a nastavovat dynamicky bez zásahu člověka jednotlivé mechanické parametry linky.

Existují sice výrobní linky schopné produkovat knihy ve velkém množství o velké taktovací frekvenci (závislá na nejpomalejším stroji v systému, řádově však jednotky tisíc taktů[1] za hodinu), ale tyto relativně velké systémy nejsou schopné ekonomicky vyrábět knihy o nízkém či velmi nízkém nákladu (v extrémním případě až o jednotkovém nákladu) a to z důvodu dlouhých nastavovacích časů.

Jak již bylo dříve zmíněno, v současné době se mění těžiště vývoje v oboru produkce knih. Hlavní důraz není již kladen na vývoj neustále rychlejších a produktivnějších systémů, které jsou schopny vyrábět větší množství knih za hodinu s menšími náklady, ale snahou je realizovat

[1] Takt = počet knih vyrobený za určité časové období. Běžně se udává buď počet taktů za hodinu (velké náklady) či počet taktů za minutu (malé náklady)

flexibilní systémy schopné produktivně a efektivně, tedy ekonomicky, vyrábět knihy s velmi malým nákladem.

Realizace těchto systémů představuje komplexní úlohu, pro jejíž úspěšné řešení je nutná důkladná znalost a pochopení problematiky jednotlivých dílčích úloh z rozdílných oblastí. Pro usnadnění návrhu a optimalizaci parametrů jsou proto dále podrobněji rozebrány hlavní oblasti dané problematiky.

V kapitole 2.4 je popsána obecná problematika In-line systémů s ohledem na výrobu malonákladových knih a přenosu dat k tomu potřebných.

Navrhování jakéhokoli výrobního systému není možné bez důkladné znalosti charakteru prováděného procesu – proto je v kapitole 2.5 nastíněna problematika průmyslové vazby knih.

V kapitole 2.6 jsou alespoň hrubě uvedeny charakteristiky průmyslových komunikačních systémů používaných k výměně dat a k řízení procesů.

V kapitole 2.7 jsou pak popsány konkrétní vybrané typy průmyslových sběrnic, které se jeví jako nejvhodnější pro budoucí realizaci řešené výrobní linky.

## *2.4 In-line systémy*

### 2.4.1 Přechod od systémů pro velké náklady na In-line systémy (pro výrobu malonákladových knih)

Doposud velmi rozšířené stroje pro tisk a další zpracování knih produkují stejné knihy ve velkém množství. V tomto případě jsou potřebné dlouhé nastavovací časy výrobní linky (cca 45 minut, vyžadující přímý lidský dohled a má-li se nastavovací čas zkrátit na minimum pak i notnou řádku zkušeností s příslušným procesem na patřičném stroji)

Dlouhé nastavovací časy jsou ale také akceptovány, neboť po úspěšném nastavení následuje výroba velkého počtu knih, takže jednotkové náklady na kus zůstávají nízké.

Tento postup je ale u malých nákladů neekonomický a u nejmenších nákladů (náklady až 10 výtisků či ještě méně) již finančně neúnosný, takže nejmenší náklady není možno průmyslově vyrábět.

Proto požadavek po nejmenších nákladech vyžaduje plně automatizované linky (na dokončovací operace) nové koncepce, jelikož jen tyto jsou schopny ekonomické výroby.

### 2.4.2 Systémy pro malonákladové knihy

Pro produkci nejmenších nákladů je tedy požadována zcela jiná výrobní strategie. Na rozdíl od klasických linek, kde tisk velkého počtu stejných stránek je realizován klasickými metodami (například ofsetovým tiskem), u malonákladových zakázek je tisk realizován digitálními produkčními systémy ve spojení se systémy na dokončující zpracování do šité nebo lepené vazby.

Individuální kniha, na konkrétní požadavek digitálně vytištěná dle osobních představ a zájmů čtenáře na základě předem daných údajů o obsahu a v jednotkovém nákladu, je konceptem nového typu výroby knih, často nazývaného Print On Demand, respektive Book On Demand.

Tento koncept představující potřebu kompletace in-line[1] systému pro rychlé zpracování malonákladových zakázek do šité nebo lepené vazby byl a je příčinou vývoje knihařských zařízení napojitelných za digitální tiskové systémy, protože zmíněná výrobní strategie klade vyšší nároky na vlastnosti systémů pro dokončující operace.

[1] In line systémy jsou takové systémy, které nevyžadují žádné mezi uskladnění produktů, ale produkt prochází kontinuálně jednotlivými operacemi v celém systému.

Samozřejmostí takovéhoto in-line systému je komplexní zřetězení (strojově a logisticky) a plná automatizace všech kroků produkce. Hlavním hlediskem je ale délka nastavovacích časů mezi jednotlivými (různými) náklady. Zařízení na dokončovací operace sice existují již dlouhou dobu, ovšem nastavení je namáhavé, zdlouhavé a vyplatí se tedy jen u velkého počtu kusů.

Nastavení konvenční dokončovací výrobní linky, tak jak je nasazována pro výrobu velkých nákladů, vyžaduje zpravidla průměrně 45 minut, během kterých vznikne makulatura asi 2% následně vyrobeného nákladu. V důsledku požadavků Book On Demand principu je kombinace takovéto linky s digitálním tiskem nemyslitelná.

V linii spojené dokončovací stroje nového typu musejí být v konečné fázi vývoje schopny flexibilně reagovat na odlišné zakázky (knihy) takovým způsobem, že při plném taktu linky mohou po sobě následovat dvě naprosto odlišné knihy. To je možné jen tehdy, pokud nastavovací časy trvají téměř nulovou dobu.

Nastavení linky na novou (odlišnou) zakázku tak musí probíhat kontinuálně při nominální provozní rychlosti (provozním taktu) linky. V této souvislosti se lze hovořit o takzvané letmé změně zakázky (německy fliegender Auftragwechsel).

Tyto podmínky kladou nové požadavky na akční členy systému. Pro nastavení polohy je třeba použít motory s vysokou rychlostí posuvu (servo pohony) a zároveň je třeba implementovat plně automatické řízení a kontrolu procesu (s odpovídajícími senzory).

### 2.4.3 Přenos dat

Aby bylo možno provést takovouto plně automatizovanou výrobu celé knihy, je zapotřebí komunikace mezi jednotlivými stroji v in-line systému. Toho je možno docílit pouze za podmínky, že všechny stroje jsou propojeny, neboli může probíhat přenos informací o jednotlivých procesních stavech. Informační tok musí být ale nejen ve smyslu dotazování na hodnoty jednotlivých procesních veličin, ale musí zde existovat i možnost ovlivnění hodnot v průběhu doby, aby bylo možno provést například automatické nastavení cesty (kanálů) nebo řešit případné detekované chyby (výskyt zmetku).

Takovýto systém pro řízení je možno navrhnout například ve formě nadřazeného řídicího počítače (průmyslové PC - IPC) a programovatelných automatů (PA), kde nadřazené IPC je propojeno s jednotlivými komunikačními moduly programovatelných automatů.

Způsob, jakým je realizováno propojení (přenos informací) pak může být různý. V principu dochází k výměně dat mezi jednotlivými komponenty (programovatelnými automaty a IPC) systému. Dle předem definovaného modelu (programu) pak dochází k řízení všech procesů (strojů).

## *2.5 Průmyslová výroba knih*

Nutným předpokladem pro úspěšně realizovanou automatizaci jakéhokoli procesu je důkladná znalost celého procesu a to do té míry, že navrhované řešení je schopno reagovat i na zdánlivě nepředvídatelné varianty, popřípadě že implementace nepředvídané varianty do hotového systému není příliš strastiplná. Výskyt konfliktních variant způsobuje totiž mnohdy značné potíže.

Proto je v další části stručně uvedena problematika výroby knihy se zaměřením na takzvanou *dokončovací výrobu*, která představuje procesy probíhající fázi tisku výroby knihy.

Pojednání o fázi tisku knihy je s ohledem na téma práce a dříve popsané použití digitálního tisku zredukováno na minimum. To reprezentuje informace o výstupu z digitálního tiskařského stroje. Ty jsou však pro výsledný produkt velmi důležité.

Pod pojmem výstupu z tiskárny je totiž nutno rozumět jednak fyzický výstup - potištěné listy, ale také výstupní informace o tom, jaký list byl právě vytištěn, jak byl vytištěn a celá řada dal-

ších informací. (Problematika výstupu tiskárny však není přímo předmětem této práce, řešení je ale realizováno způsobem umožňujícím budoucí napojení na digitální tiskový stroj.)

### 2.5.1 Rozdělení dokončovací výroby

Dokončovací výrobu obecně lze rozdělit do tří hlavních oblastí:

- *jednoduché knihařské práce,*
- *zhotovování knižních vazeb a*
- *speciální knihařské práce*.

Z důvodů zaměření této práce je dále podrobněji popsána pouze oblast zhotovování knižních vazeb. Existují tři typy průmyslových vazeb - měkké vazby, tuhé vazby a speciální vazby.

*Měkké vazby* vzniknou spojením sešitého nebo lepeného knižního bloku s papírovou, kartónovou, plátěnou nebo laminovanou obálkou.

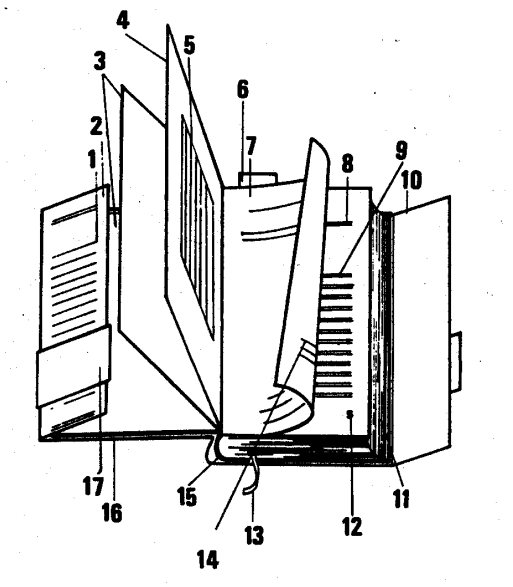
*Tuhé vazby* vzniknou spojením sešitého nebo lepeného knižního bloku s tuhými knižními deskami, které mohou mít kombinovaný nebo nekombinovaný potah, resp. s knižními deskami zhotovenými z plastu.

*Speciální vazby* pak představují různé způsoby spojení listů a složek do jednoho celku (spirálou, hřebenem z plastů a podobně) a prostorových obálek.

Před podrobnějším popisem procesu vazby knihy je účelné nejdříve připomenout, jaké jednotlivé části tvoří knihu spolu s jejich stručnou funkcí.

### 2.5.2 Kniha a její části

Během dlouhodobého vývoje se postupně víceméně ustálil tvar knihy a její části. Přesto ale neexistuje pouze jeden typ knihy. Většina důležitých částí je znázorněna na obr. 2.5-1.




*Obr. 2.5-1 Kniha a její části.*

*1-anotace, 2-záložka přebalu, 3-předsádka, 4-patitul, 5-protitulní obrázek, 6-papírová záložka 7-titulní list, 8-záhlaví, 9-text v knize, 10-přebal, 11-ořízka, 12-stránkové číslo, 13-stužková záložka, 14-vydavatelský záznam, 15-kapitálek, 16-knižní desky, 17-páska*

### 2.5.3 Vazba knihy

Do příchodu potřeby individuálně vyrábět knihy průmyslově On-Demand, bylo možno principielně rozlišovat dva typy vazeb.

Prvním typem byla vazba ruční nebo umělecká, která se vyznačovala více nebo méně individuálními prácemi na zakázku, a druhým typem pak vazba průmyslová.

V průmyslové vazbě se knihy zhotovují hromadně (sériově) a je charakterizována určitou standardností výkonů, jejichž výsledkem je kniha jako průmyslový výrobek.

Kvůli zmíněné potřebě vyrábět průmyslově i malonákladové knihy dochází k určitému prolínání těchto dvou typů.

Avšak požadavek průmyslové vazby spočívající v minimální nutnosti zásahu lidskou rukou do procesu zůstává stále platný, naopak je s ohledem na ekonomičnost provozu o to více kritický.

To přináší celou řadu problémů, které kromě samotné automatizační úlohy pramení i z ne zcela jasně definovaného standardu knihy.

V České republice je výroba knižních vazeb standardizována na základě původní normy ON 88 3750, která byla novelizována. Přesto výklad normy není zcela jednoznačný a především její znalost odbornou veřejností není dokonalá. Normy v ostatních státech jsou pak přirozeně jiné (pokud existují) a určení, jak má která vazba vypadat, je založeno, spíše než na jasné definici, na základě praktických zkušeností a požadavcích vydavatele.

Přesto lze tento postup výroby knihy obecně charakterizovat. Kniha vzniká v závěrečné fázi její výroby vykonáním několika technologických procesů (dokončovací výrobní fáze). Nejdříve se potištěné archy skládají na knižní složky. Ty po snesení vytvoří knižní komplet, který po spojení ve hřbetě nazýváme knižním blokem. Odděleně se zhotovují knižní desky, které se s knižním blokem spojí pomocí předsádky.

To vše probíhá pomocí speciálních strojů. Setkáváme se zde s prvky a pojmy, které je třeba pro snazší pochopení celého procesu alespoň stručně vysvětlit. Jednotlivé části hotové tuhé vazby jsou znázorněny na obr. 2.5-2, její vnitřní struktura pak ještě na obr. 2.5-3. Dále je tedy uveden stručný popis pojmů (řazených podle technologického postupu) výroby knihy.

*Knižní arch* - potištěný arch papíru, který se při dokončovacím zpracování skládá na knižní složky

*Knižní složka* - potištěný arch papíru složený jedním nebo několika lomy na předepsaný formát knihy.

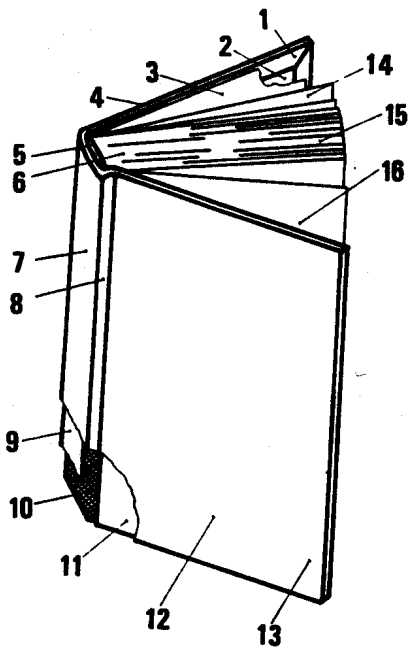
*Předsádka* - dvojlist potištěného nebo nepotištěného papíru určený ke spojení knižního bloku s knižními deskami a na ochranu prvního a posledního listu knihy; lze ho nahradit vnějším listem první a poslední složky (nepravá předsádka).

*Příloha* - ilustrace, tabulka, mapa nebo jiný názorný doplněk knihy, tištěný odlišně nebo na jiném papíru než textová část a zařazený do textové části nebo za ní, např. lepením, zasunutím, vložením pod pásku, popřípadě do kapsy na přídeští knihy.

*Knižní komplet* - úplná sestava snesených listů, složek, popřípadě i příloh, určená ke zhotovení knižního bloku.

*Knižní blok* - knižní komplet spojený ve hřbetu lepením nebo šitím, tvoří vnitřní část knihy.

*Ořízka* - boční strany knižního bloku po ořezání.




*Obr. 2.5-2 Tuhá vazba a její části*

*1 - záložka potahu, 2 - přídeští, 3-zadní předsádka, 4-zadní knižní deska, 5-kapitálek, 6-ořízka, 7-hřbet knižních desek, 8-drážka, 9-hřbetník, 10-knihařský gáz, 11-přířez přední knižní desky, 12-přední knižní deska, 13-potahový materiál, 14-předsádka, 15-knižní blok, 16-předsádka*

*Kapitálek* - lemovka (obruba), která se lepí na norní a dolní okraj oříznutého knižního bloku (zpevňuje i zdobí knižní blok)

*Dutinka* – papírová vložka přilepená ke hřbetu knižního bloku a vnitřní straně hřbetníku knižních desek. Zabezpečuje pevnější spojení knižního bloku s knižními deskami.

*Knižní desky* - obal knižního bloku zhotovený ze dvou lepenkových přířezů a ze hřbetníku (proužek lepenky nebo kartónu mezi přední a zadní deskou), potažený papírem, plátnem nebo jiným materiálem.

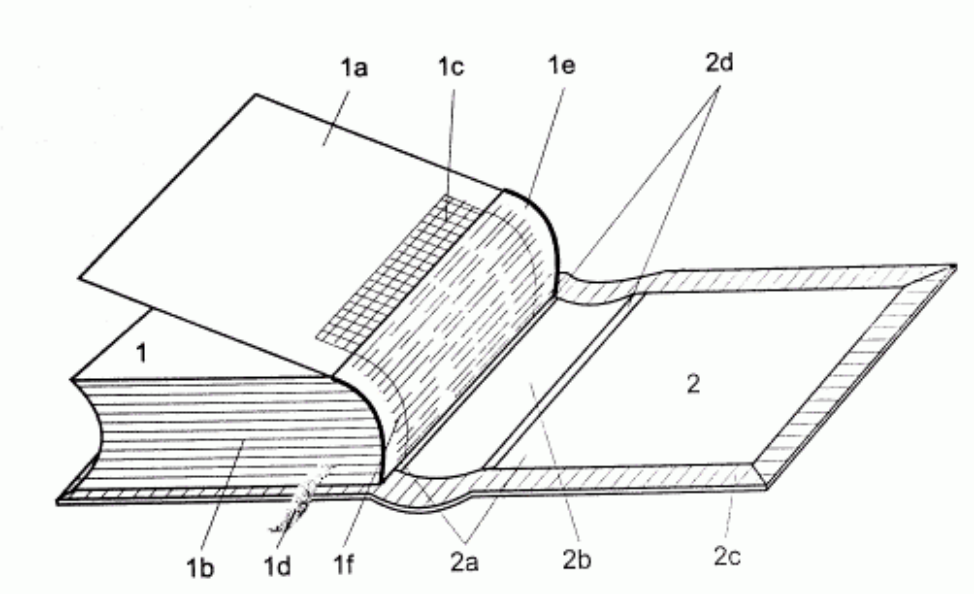
*Přídeští* - vnitřní strany knižních desek, na kterých je přilepená předsádka

*Knižní hřbet* - část knižní vazby, kterou tvoří hřbet knižního bloku a knižních desek.

*Hřbetní drážka* - žlábek tvarovaný tlakem nástroje mezi hřbetníkem a přířezem lepenky potažených knižních desek.

*Obálka* - obal brožury (měkká vazba) z papíru nebo kartónu, může být potištěný nebo nepotištěný.

*Přebal knihy* - obal knihy z papíru, fólie nebo jiného materiálu s dovnitř přehnutými okrajovými pásy (záložkami).




*1 knižní blok*
*1a předsádka*
*1b složené potištěné listy, snesené, sešité (nebo lepené) a ořezané*
*1c knihařský gáz*
*1d záložková stužka*
*1e kapitálek*
*1f dutinka*
*2 knižní desky*
*2a lepenkové přířezy*
*2b hřbetník*
*2c záložka potahu knižních desek*
*2d hřbetní drážka*

*Obr. 2.5-3 Vnitřní struktura tuhé vazby*

2.5.3.1.1 Technologický postup

Je zřejmé že náročnost výroby je odlišná u měkkých a u tuhých vazeb. Kniha s tuhou vazbou je obecně kvalitnější, ale její výroba je zároveň složitější a automatizace celého procesu vyžaduje řešení komplexnějších problémů

Pro lepší představu je dále rozebrán proces výroby knihy s měkkou vazbou (Softcover) a tuhou vazbou (Hardcover). Základním rozdílem mezi měkkou a tuhou vazbou je druh použitého materiálu na obálku, dále je pak odlišné spojení obálky s knižním blokem .

Měkká vazba (brožura) je spojení šitého nebo lepeného knižního bloku s papírovou, kartónovou, plátěnou popřípadě laminovanou obálkou. V rámci této skupiny vazeb je možno zhotovit následující typy brožur:

- sešitová měkká vazba
- lepená měkká vazba
- bloková měkká vazba
- šitá měkká vazba

Tuhá vazba je spojení knižního bloku s knižními deskami z tuhé lepenky nebo plastu. V rámci této skupiny se rozlišují:

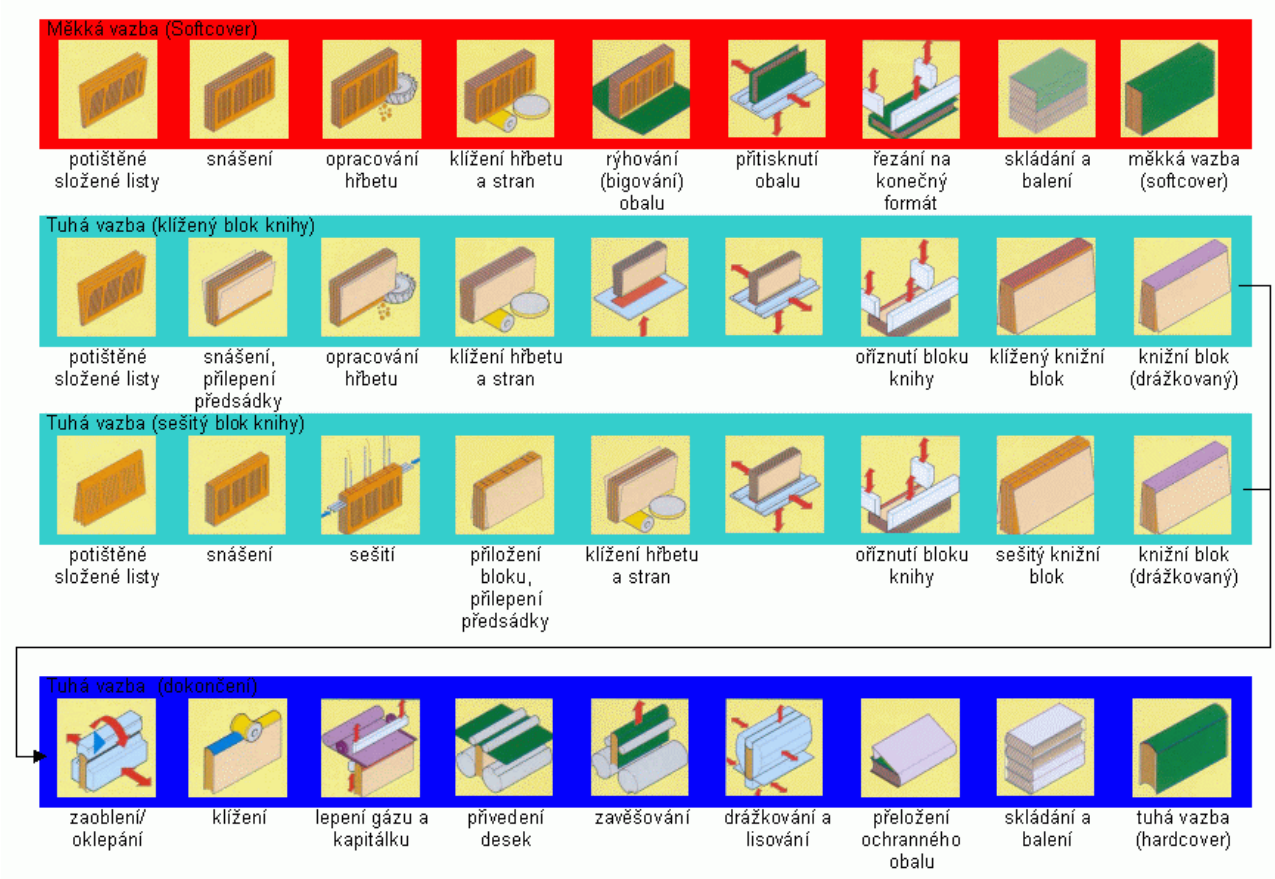
- vazba s kombinovaným potahem (poloplátěná vazba s tuhými knižními deskami potaženými papírem, hřbet je potažen plátnem nebo jiným vhodným materiálem)
- vazba s nekombinovaným potahem (tuhé knižní desky jsou celé potaženy plátnem

- tuhá vazba s laminovaným potahem
- tuhá vazba s papírovým potahem
- tuhá vazba s potahem z jiného materiálu
- vazba z plastu

Zjednodušené porovnání výroby měkké a tuhé vazby znázorňuje obr. 2.5-4.


*Obr. 2.5-4 Porovnání technologie výroby měkké a tuhé vazby*

Obr. 2.5-5 pak graficky pomocí schematických nákresů znázorňuje technologický postup při výrobě měkké a tuhé vazby (konkrétně měkké vazby lepené a tuhé vazby jak lepené tak šité).

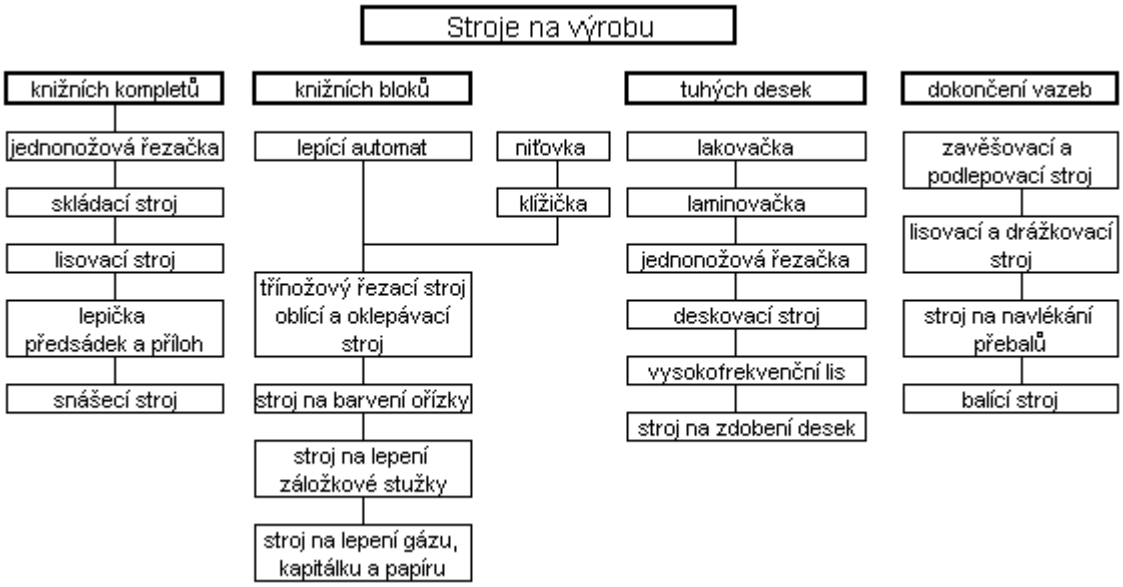



*Obr. 2.5-5 Přehled technologického postupu vazby knihy*

### 2.5.4 Přehled strojů používaných v dokončovací výrobě

Na obr. 2.5-6 je naznačeno dělení strojů používaných v dokončovací výrobě. Dělení je provedeno na základě toho, na jaké části výroby knihy se stroje podílejí. Patřičnou kombinací odpovídající technologickému postupu uvedeném na obr. 2.5-5 je možné stroje seřadit do výrobní linky.

Nutným předpokladem pro zařazení do řetězce linky je kromě schopnosti komunikovat s řídicím systémem také ošetření logistického rozhraní pro dopravu produktu na výstupu jednoho a vstupu následujícího stroje. Při konstrukci takovýchto strojů je pak nutno pamatovat na vstupní a výstupní pásy. Pokud mají být použity již existující stroje, je mnohdy nutno zařadit mezi jednotlivé stroje zařízení umožňující plynulý přechod z patřičného výstupu na následující vstup stroje dalšího.




*Obr. 2.5-6 Přehled strojů používaných při dokončovací výrobě*

## *2.6 Průmyslové komunikační systémy*

Ač popsaný proces výroby knihy je svým způsobem specifický, princip řízení má mnoho společného s ostatními odvětvími. Řízení všech těchto odvětví je možno shrnout pod pojem řízení technologických procesů.

Speciální podskupinou je pak řízení technologických linek ve výrobě, kde se setkáváme s řadou na první pohled odlišných procesů při výrobě, např. výroba chemických, strojírenských, stavebních, potravinářských, farmaceutických, automobilových a v neposlední řadě i polygrafických produktů. Poslední dva pak bývají považovány za nejkomplexnější. Všechny tyto procesy ale mají společné některé základní charakteristiky:

- technologická linka není centralizována do jednoho “boxu”, ale je rozložena např. po celé výrobní hale
- pro optimální řízení procesu je nutné rychlé a bezporuchové měření všech požadovaných veličin procesu na rozlehlém prostoru
- rychlé a spolehlivé musí být rovněž ovládání akčních členů
- cyklus zpracování naměřených signálů musí proběhnout v určitém stanoveném časovém intervalu.

Tyto vybrané charakteristiky ve svém důsledku v podstatě vyžadují přenos informace o procesních parametrech mezi jednotlivými zařízeními, realizované určitým způsobem, zaručující splnění vyčtených požadavků. Ať už se jedná pouze o přenos informace typu zapnuto/vypnuto či komplexnější zprávy, je možno tuto výměnu shrnout pod pojem přenos dat.

*Přenosy dat* u technologického systému je v současné době možné z velké části realizovat podle požadavků technologie pomocí průmyslových komunikačních sběrnic. Tato technologie se sériovým přenosem dat je kompromisem mezi cenou zařízení a rychlostí přenosu - tedy mezi sériovou a paralelní sběrnicí. Paralelní přenos na vzdálenosti řádově stovky metrů je finančně neúnosný.

Měřicí a řídicí systémy jsou, na rozdíl od počítačových sítí, modulární, sestavené pro konkrétní aplikaci s jednoúčelovým aplikačním programovým vybavením.

Pod pojmem průmyslový řídicí systém rozumíme měřicí, řídicí a informační systém určený pro provoz v prostředí, které z hlediska elektromagnetické kompatibility, vnějších vlivů a způsobu nasazení odpovídá běžným podmínkám průmyslové praxe.

V další části je nejprve popsán vývoj sběrnic a jejich struktura. Poté je uveden přehled jednotlivých sběrnic doplněný konkrétním popisem vybraných sběrnic.

### 2.6.1 Vývoj průmyslových sběrnic a jejich hierarchické dělení (vnější)

#### *2.6.1.1 Vývoj*

Řešení problému propojení většího množství spolupracujících zařízení dohromady a zároveň splnění výše uvedených požadavků na topologii, řízení, měření, rychlost, spolehlivost přenosu dat a časovou odezvu (schopnost práce v reálném čase) není jednoduché. Ideálním výsledkem z hlediska uživatele by byl takový typ průmyslové sběrnice, který by představoval rozsáhlý systém standardizovaný do té míry, že by umožňoval uplatnit různá zařízení od různých výrobců.

V současné době je na trhu velké množství průmyslových komunikačních systémů od různých výrobců. Přes jisté snahy o standardizaci, které ale přišly se zpožděním, je jejich vzájemná kompatibilita spíše problematická a to z hned několika důvodů.

Prvním důvodem jsou *rozdílné požadavky na vlastnosti* průmyslových sběrnic. Ty jsou značně závislé, na *jakém místě* ve struktuře celého řídicího systému *mají být nasazeny*. Mimo jiné nároky na rychlost přenosu, objem a zabezpečení přenášených dat jsou rozdílné, má-li příslušná sběrnice propojit programovatelný automat se senzory a akčními členy nebo naopak s nadřazenými vrstvami řízení. Současně *oblast nasazení* dále poněkud modifikuje výsledné požadované vlastnosti. Ty jsou tedy mírně odlišné, jedná-li se o řízení strojírenské výroby a montáže, regulaci chemických procesů, automatizaci budov a podobně.

Kromě zmíněných rozdílných požadavků je dalším důvodem velkého počtu sběrnic i *způsob jejich vzniku*. Sběrnice byly vyvíjeny nezávisle různými výrobci specializujícími se na různé aplikace v důsledku akutní potřeby řešení problému rostoucí automatizace, kdy mnohdy staré řešení propojení již bylo ekonomicky neúnosné či nemyslitelné. Určité snahy o normalizaci se pak dlouho uplatňovaly, pokud vůbec, nanejvýš na národní a nikoliv na mezinárodní bázi. Při takovémto vývoji tak vzniklo velké množství rozdílných sběrnic nejen proto, že rozdílné požadavky kladené na komunikaci po průmyslových sběrnicích lze navíc ještě zabezpečit různými způsoby, ale i z toho důvodu, že výrazně odlišná může být i představa, *jaký úkol by měla sběrnice plnit*. V zásadě lze rozlišit dva typy přístupů.

a) Průmyslová komunikační sběrnice je pouze prostředkem umožňujícím zjednodušit komunikaci mezi jednotlivými zařízeními ve víceméně elektrotechnickém slova smyslu, což představuje podstatné snížení počtu propojovacích vodičů a zjednodušení kabeláže.
b) Průmyslová komunikační sběrnice je skutečnou páteří distribuovaného řídicího systému určeného pro práci v reálném čase.

Z výše uvedených důvodů problematické kompatibility je patrné, že existence jednoho univerzálního standardu průmyslové sběrnice je nereálná a také nemá smysl. Přesto existuje několik oblastí, pro které by bylo možno jasně definovat požadavky na sběrnici a jakou úlohu by měla

plnit. Pro jednotlivé oblasti by pak bylo teoreticky možné stanovit jednotný standard, který by zaručoval kompatibilitu všech zařízení používaných v tomto subsystému. To se však v současné době jeví více než problematické, neb zde mimo jiné důležitou roli hraje také marketingová strategie jednotlivých výrobců. Jejich přirozenou snahou je vázání zákazníka na své vlastní výrobky. Proto situace, kdy určitý nabízený systém je bezproblematicky kompatibilní pouze v rámci výrobků jedné firmy popřípadě sdružení firem, je vcelku výhodná.

Přesto v poslední době standardizace dosáhla relativně dobré úrovně. Pokud není možné komunikovat po stejné sběrnici, je možno většinou objednat u výrobců alespoň určitý modul, který umožní komunikaci mezi odlišnými standardy sběrnic.

#### *2.6.1.2 Dělení*

Před přechodem z obecné roviny do roviny konkrétního popisu průmyslových sběrnic je ještě důležité zdůraznit strukturu a hierarchii komunikačního řetězce v průmyslovém výrobním procesu, neb sběrnice jsou patrně jen jednou z více součástí celého systému.

Velmi zhruba lze rozlišit trojici hierarchicky uspořádaných úrovní řízení:

- úroveň řízení výroby
- úroveň řízení procesu
- úroveň bezprostředního řízení

V *úrovni řízení výroby* jsou soustřeďovány, archivovány a zpracovávány důležité údaje o technologickém procesu a na základě jejich vyhodnocení je možné výrobní proces plánovat a optimalizovat z hlediska kvality výroby, ekonomické náročnosti, materiálových úspor i dalších obdobných parametrů. Sem spadá i logistická podpora výroby, její koordinace se subdodavateli a požadavky odběratelů. Všechny tyto činnosti vyžadují přenos velkého množství dat. Objem dat v jednom přenosu se proto pohybuje v řádech jednotek Mbyte. Požadované doby odezvy při požadavku jsou však poměrně dlouhé. Komunikační sítě v této rovině bývají označovány LAN (Local Area Network), WAN (Wide Area Network) popřípadě GAN (Global Area Network).

Druhá vrstva - *úroveň řízení procesu* - provádí automatizované řízení celého technologického procesu popřípadě montážní či výrobní linky a koordinuje a optimalizuje činnost regulátorů umístěných ve třetí vrstvě, které pak už mají za úkol regulovat jednotlivé stroje a zařízení. Na této úrovní se provádí také vizualizace dat z procesu a jsou možné operátorské zásahy do nejdůležitějších veličin a parametrů procesu. Proto bývá také někdy označována jako operátorská úroveň. Objem přenášených dat je již menší, podstatně však vrůstají nároky na rychlost. Požadovaná data musí obvykle být k dispozici nejpozději se zpožděním desetin sekundy či nanejvýše jednotek sekund. Komunikace na této rovině se odehrává již v relativně menší oblasti a jedná se tedy většinou o lokální síť. Vzhledem k tomu, že zařízení této úrovně však pracují v náročnějších podmínkách, než je obvyklé kancelářské prostředí, bývají místo standardního Ethernetu, který je obvyklým základem kancelářských lokálních sítí, často používány jeho modifikace, vylepšené pro účely průmyslového použití (odolnost proti rušení, garantovaná doba odezvy) a shrnované pod obecný název průmyslový Ethernet.

V *úrovni bezprostředního řízení* se již nacházejí přístroje, které řídí jednotlivé stroje a dílčí procesy, to znamená především regulátory, programovatelné automaty a jejich senzory a akční členy. Objem dat přenášených při jednom přenosu výrazně klesá až na jednotky byte, podstatně však vzrůstají požadavky na rychlost odezvy, která může být požadována až v jednotkách ms. Z hlediska rozsahu by sítě používané na této úrovni bylo možno označit také za lokální. Vzhledem k specifickým požadavkům na spolehlivost přenosu v prostředí s rušením, schopnost práce v reálném čase a podobně, které jsou na ně kladeny, vyčleňují se jako samostatná kategorie a nazývají se průmyslové komunikační sítě (FAN - Field Area Network).

Nastíněné členění je schematické a hranice jednotlivých úrovní nejsou ostré, stejně jako dochází k prolínání úrovní, překrývají se i aplikační oblasti jednotlivých typů sítí. Některé typy komunikační sítě třetí úrovně se tak používají i k plnění některých úkolů z druhé vrstvy. Naopak například průmyslový Ethernet je již používán ve třetí úrovni a částečně vytlačuje stávající průmyslové sítě z jejich pozic, především v oblasti přenosu většího objemu dat.

#### *2.6.1.3 Sběrnice a automatická linka pro výrobu knih*

Vzhledem k zaměření této práce je dále podrobněji kladen důraz na druhou a třetí úroveň. Je zřejmé, že v úloze plně automatického procesu výroby knih o malém nákladu, kde po sobě následují různé knihy, je třeba komunikace nejen v oblasti rychlého přenosu dat mezi programovatelnými automaty s senzory a akčními členy (třetí vrstva), ale i v oblasti přenosu instrukcí obsahujících údaje o jednotlivých vyráběných knihách pro programovatelné automaty (vrstva druhá).

Při výběru některého z řady typů průmyslových sběrnic je nutné znát krajní podmínky pro propojení řídicích stanic a periferií a podmínky komunikace z hlediska využití přenosového kanálu. S jednotlivými funkčními jednotkami je možné komunikovat zcela pravidelně (většinou v krátkých časových intervalech) nebo podřízené subsystémy pracují v podstatě samostatně a přenášejí se pouze již nějakým způsobem komprimované údaje. Rozhodnutí o způsobu komunikace, struktuře a typu průmyslové sběrnice by tak měla předcházet *důkladná analýza celého technologického procesu*.

Kromě znalosti technologického procesu je také třeba znát vlastnosti a použitelnost jednotlivých typů sběrnic, tak aby výsledná volba byla optimální, a to nejen z hlediska technického řešení, ale i finanční náročnosti a v neposlední řadě i možnosti dalšího vývoje respektive rozšiřitelnosti a vylepšení. Proto jsou v dalších kapitolách podrobně popsány a následně zhodnoceny vybrané sběrnice s ohledem na použitelnost v řešeném systému. Na tomto místě je však třeba podotknout, že z již uvedených důvodů kompatibility je výběr mnohdy omezený. Výběrem systému z určité skupiny výrobců (výrobce) je možno eliminovat možné problémy s uzpůsobováním kvůli kompatibilitě. Skupina výrobků je pak mnohdy stejně již dopředu určena dříve realizovanými projekty. Orientace na jednu skupinu přináší totiž mnohé výhody jak finanční tak z oblasti know-how a další.

Ještě před orientací na jednotlivé sběrnice je účelné vymezit o objasnit některé pojmy dále používané, stejně tak jako doplnit informace o vnitřní hierarchické struktuře sběrnic, obsažené v referenčním modelu ISO/OSI.

### 2.6.2 Vymezení pojmů

Průmyslový komunikační systém – průmyslová sběrnice - Fieldbus

Pro průmyslové komunikační systémy se v angličtině používá krátkého vžitého výrazu *fieldbus*. Český adekvátní výraz neexistuje. Používá se výrazů průmyslové sběrnice, distribuované systémy nebo průmyslové komunikační systémy. Ani jeden z těchto názvů nevystihuje bohužel tak stručně a výstižně podstatu diskutovaných systémů jako výraz fieldbus.

Protokol, standard

Protokolem rozumíme způsob řízení komunikačního systému včetně definice přenosového protokolu. Standard sběrnice specifikuje podrobnou aplikaci ISO/OSI protokolu.

Sběrnice

Fyzické propojení několika zařízení podle standardu ISO/OSI.

Zařízení, stanice, uzel, modul, účastník

Pro rychlou a spolehlivou komunikaci signálů mezi senzory, akčními členy, řídicí jednotkou a dalšími prvky je důležité navrhnout zpracování naměřených signálů v blízkosti senzoru jednoči-

povým mikrokontrolérem. Tuto funkci včetně předzpracování signálu a zajištění komunikace s řídicí jednotkou nebo jednotkou, která požaduje tyto informace, zabezpečuje *stanice, zařízení, modul, účastník* nebo také *uzel.* Všechny tyto názvy se používají pro popis části distribuovaného systému.

Master/Slave

Způsob řízení, kdy jedna řídicí stanice (master) řídí několik podřízených stanic (Slave). Komunikace mezi dvěma podřízenými stanicemi se realizuje přes Master. Jestliže systém je složen z více zařízení Master, hovoříme o Multimaster.

Producer/Consumer

Způsob přenosu dat mezi jejich zdrojem a příjemcem, bez ohledu na způsob řízení, zda zdrojem je např. Master nebo Slave.

### 2.6.3 Vnitřní hierarchická struktura (referenční model ISO/OSI)

Kromě vnější struktury, do níž průmyslové sběrnice zařazujeme, však lze a je účelné rozlišit i jistou vnitřní strukturu činností, které musí probíhat, aby se mohl uskutečnit přenos dat mezi dvěma zařízeními často velmi odlišné povahy. Pro úspěšný přenos dat je nutné stanovit zásady pro vzájemnou komunikaci procesorů v distribuovaných systémech řízení i mezi těmito systémy v integrovaných sítích.

Specifikaci těchto zásad v normě si vynutily problémy se vzájemnou nekompatibilitou jednotlivých druhů sítí a komunikačních protokolů. V roce 1981 tak bylo nejdříve vydáno doporučení IEEE 802, které v obecné podobě určuje způsob komunikace mezi zařízeními od úrovně fyzického připojení až po způsob pohybu rámců (přenosových protokolů) v síti.

Doporučení až do úrovně specifikování aplikace programů pak v roce 1983 vydala organizace International Standard Organization (ISO) pod názvem referenční model OSI (Reference Model for Open System Interconnection) jako normu ISO 7498. IEEE 802 pokrývá pouze 3 nejnižší vrstvy OSI protokolu. OSI specifikuje soubor standardů pro výměnu informací mezi systémy, které jsou vůči sobě vzájemně otevřené, tj. respektují stejné normy. Model OSI je modulární a umožňuje nové aplikace nebo služby bez změny struktury modelu.

Model znázorňuje jakým způsobem se vysílaná zpráva (například požadavek na zapnutí stroje, čtení hodnoty snímače a podobně) postupně kóduje, doplňuje o zabezpečovací bity atd. až vznikne nakonec elektrický, optický nebo jiný signál, který je možné po odpovídajícím médiu vyslat a na přijímací straně potom v opačném sledu převést na zprávu, kterou bude přijímací zařízení schopné správně interpretovat. Činnosti, které během tohoto procesu mohou probíhat, jsou v modelu rozčleněny do sedmi vrstev, z nichž každá má své specifické úkoly. Vzájemná spolupráce by se měla odehrávat přes definovaná rozhraní a mělo by tedy být možné, aby komunikace probíhala, přestože jednotlivé vrstvy byly vytvořeny různými výrobci a různým způsobem. Tímto způsobem by měla být zaručena zmíněná otevřenost systému.

OSI tedy zcela obecným způsobem definuje způsob komunikace jak mezi jednotlivými subsystémy informačního systému, tak i mezi jednotlivými vrstvami komunikačního modelu. Každá vrstva definuje vlastnosti obou svých rozhraní specifikací služeb požadovaných od nižší vrstvy a specifikací služeb předávaných vrstvě vyšší.

OSI model je svým způsobem maximalistický. Přestože většina průmyslových, stejně jako ostatních, sítí musí vykonávat ty činnosti, které tento model popisuje, ne vždy je jejich struktura vytvořena takovým způsobem, aby bylo možno jasně vymezit sedm vrstev modelu. Specielně v případě průmyslových sítí určených pro práci v reálném čase či sítí s velkou a pokud možno zaručenou rychlostí odezvy by to bylo neúčelné a zbytečně komplikované, protože rozdělení přenosu do sedmi vrstev činností by průběh komunikace nutně zpomalilo.

Proto často struktura průmyslových komunikačních sítí je založena na sloučení několika vrstev modelu do jedné, popřípadě dělení vrstev je jiné. Navzdory tomu je referenční model užitečnou pomůckou a usnadňuje popis a pochopení komunikace v sítích stejně jako srovnání struktur různých typů sítí. Proto je dále model OSI podrobněji rozebrán.

#### *2.6.3.1 Popis modelu OSI*

Na obr. 2.6-1 je schematicky znázorněno uspořádání jednotlivých vrstev.

Nejvyšší sedmá, takzvaná *aplikační vrstva* (Application Layer), je vrstvou aplikačních rozhraní a poskytuje služby dalším programům, které prostřednictvím sítě komunikují.

Šestá *prezentační vrstva* (Presentation Layer) zabezpečuje převod dat do formy vhodné pro přenos. Provádí tedy převody kódů do formátů dat případně i jejich kompresi a šifrování.

Pátá *relační vrstva* (Session Layer) je zodpovědná za tvorbu, udržování, synchronizace a ukončování spojení mezi jednotlivými účastníky. Někdy bývá také označována jako vrstva pro řízení komunikace.

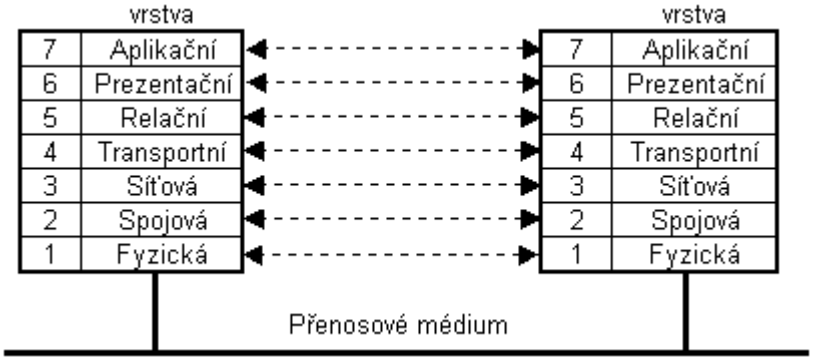
Čtvrtá *transportní vrstva* (Transport Layer) kontroluje a řídí práci nižších vrstev a zabezpečuje bezchybný přenos dat ve správné posloupnosti bez ztrát či duplikací. Umožňuje současnou komunikaci více aplikačních programů na jednom počítači v síti, provádí rozklad rozsáhlejších zpráv do paketů a naopak. Zabývá se tedy komunikací koncových uživatelů.

Třetí *síťová vrstva* (Network Layer) zabezpečuje přenos dat v síti tak, aby vyšší vrstvy mohly komunikovat, aniž by znaly strukturu sítě, její přenosové charakteristiky a podobně. Význam má především v sítích, v nichž existují alternativní přenosové cesty mezi přijímačem a vysílačem, protože jejím hlavním úkolem je tyto nejvhodnější přenosové cesty hledat a vytvářet. Bývá proto také nazývána jako zprostředkující či spojovací vrstva.

Druhá *spojová*, někdy také *linková*, *vrstva* (Data Link Layer) zabezpečuje bezchybný přenos bloků dat a lze se s ní proto někdy setkat i pod názvem zabezpečovací vrstva. Pro účely přenosu vlastní přenášená data doplňuje o synchronizační posloupnosti, adresaci, zabezpečovací kódy a další prvky, které dohromady vytváří tzv. rámec a řídí rovněž přístup k přenosovému médiu.

První a konečná *fyzická vrstva* (Physical Layer) pak zajišťuje skutečný fyzický přenos jednotlivých bitů přenášené zprávy. Na úrovni této vrstvy jsou definovány fyzické parametry sítě, jako jsou například:

- přenosové médium (kroucený dvoudrát, koaxiální kabel, optické vlákno, ...)
- přenosové rychlosti
- připojovací konektory
- logické úrovně
- topologie sítě



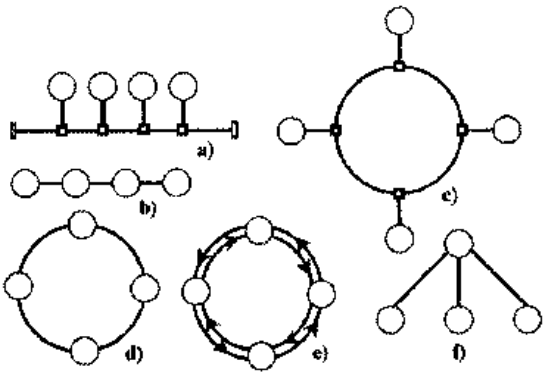

*Obr. 2.6-1 Referenční model ISO/OSI*

2.6.3.1.1 Fyzická vrstva

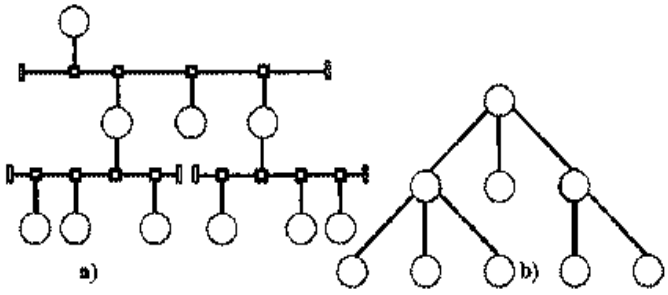
Jednou z prvních vlastností, které tato vrstva definuje je topologie. Pod tímto termínem se obvykle rozumí struktura fyzického propojení, to je způsob, jak jsou vedeny spojovací kabely v síti. Od této fyzické topologie je ale obecně nutno rozlišit tzv. signálovou topologii popisující šíření signálu po síti, která se ne vždy s fyzickou topologií musí shodovat. Kromě toho bývá ještě na rovině spojové vrstvy rozlišována takzvaná logická topologie, která u deterministických metod řízení určuje způsob spolupráce jednotlivých stanic.

Na obr. 2.6-2 jsou znázorněny základní struktury topologie jednoúrovňových sítí. Nejběžnějším typem je pasivní sběrnice - varianta a). Při větším počtu připojených zařízení je výhodnější použít aktivní sběrnice - varianta b). Dalším typem je pasivní kruh - varianta c). Častěji používaný je kruh s aktivním připojením - varianta d). Jeho modifikací vznikne takzvaný obousměrný kruh - varianta e).

Kombinací uvedených základních struktur lze vytvářet složitější struktury. Některé jsou znázorněny na obr. 2.6-3. Vícevrstvá sběrnice s pasivním připojením je schematicky znázorněna jako varianta a). Spojením několika hvězdicových podsítí vznikne strom - varianta b). Různých variant těchto vícevrstvých struktur existuje samozřejmě daleko více.




*Obr. 2.6-2 Základní struktury topologie jednoúrovňových sítí*




*Obr. 2.6-3 Některá uspořádání topologie víceúrovňových sítí*

S volbou topologie do jisté míry souvisí i volba přenosového média. Nejčastěji používanými médii jsou: *kroucený dvoudrát*, který může být stíněný (STP - Shielded Twisted Pair) nebo nestíněný (UTP - Unshielded Twisted Pair); dále *koaxiální kabel*, který má lepší stínění a je tedy odolnější proti rušení; *optická vlákna* jako alternativní přenosové médium. Na rozdíl od metalických jsou dokonale odolná proti elektromagnetickému rušení. U optických vláken ale představuje problém vyšší finanční náročnost - ani ne optických vláken samotných, ale vyšší cena nutných rozhranní a konektorů.

2.6.3.1.2 Spojová vrstva

Podle; normy IEEE 802 je spojová vrstva dále rozdělena na dvě podvrstvy - řízení přístupu k přenosovému prostředku (MAC - Media Access Control) a řízení logického spoje (LLC - Logical Link Control). Podvrstva MAC stojí blíže k fyzické vrstvě a zabezpečuje služby a funkce specifické pro daný přenosový prostředek. Vyšší podvrstva LLC poskytuje nadřazeným vrstvám služby přenosu dat a navazování spojení.

Jednou z nejdůležitějších funkcí spojové vrstvy je řízení přístupu ke sdílenému přenosovému prostředku (obvykle sběrnici), které koordinuje činnost jednotlivých vysílačů a řeší jejich případné kolize tak, aby všechny potřebné datové přenosy byly nakonec provedeny. Metody používané k tomuto účelu lze rozdělit do dvou velkých skupin:

a) metody s náhodným přístupem
b) metody s řízeným (deterministickým) přístupem. Ty lze ještě dále dělit na:

- b1) centralizované
- b2) distribuované

Nejjednodušší z metod s náhodným přístupem je postup označovaný jako metoda příposlechu nosné s mnohonásobným přístupem (CSMA - Carrier Sense Multiple Access). Stanice která chce vysílat, nejprve zkontroluje, zda sdílený přenosový prostředek není obsazen (Carrier Sense). Je-li volný, začne vysílat, je-li naopak obsazen, počká a později se pokusí o vysílání znovu (Multiple Access).

Z hlediska doby čekání lze ještě rozlišit několik podvariant této metody. První je takzvaná naléhající (persistent) CSMA, kdy stanice se pokusí o vysílání okamžitě poté, co dojde k uvolnění sběrnice. U většího počtu stanic je pravděpodobné, že o vysílaní se pokusí více stanic najednou a dojde tedy ke kolizi. Tomuto jevu zabraňuje do jisté míry nenaléhající (non-persistent) CSMA, kde stanice čeká náhodnou dobu než začne opět vysílat. To vede ale k velkému zpoždění přenosu. Určitým kompromisem je pak p-naléhající (p-persistent) CSMA, kde stanice po uvolnění kanálu začne s pravděpodobností p vysílat a s pravděpodobností 1-p odloží činnost na krátkou dobu a pak provede test obsazenosti a na základě jeho výsledku zahájí vysílání nebo dále čeká.

Všechny tyto varianty CSMA mají náhodný charakter. Stanice se může pokusit o vysílání kdykoliv a čas uplynulý od prvního pokusu o vyslání zprávy do skutečného odvysílání nelze předem stanovit. Tato druhá vlastnost omezuje použitelnost této metody v průmyslových komunikačních sítích, kde je obvykle nutná schopnost práce v reálném čase se zaručenou dobou odezvy.

Zlepšení lze dosáhnout doplněním o detekci kolizí. Jedná se o metodu CSMA/CD (Collision Detection) a pracuje například u sítě Ethernet tak, že pokud některá ze stanic zjistí, že začala vysílat zároveň s jinou, přeruší vysílání a zvláštním krátkým signálem (jamming signal) oznamujícím kolizi informuje ostatní stanice, které pak rovněž přeruší vysílání a teprve po náhodně dlouhé době se o něj pokusí znovu. Ještě lepšího chování lze dosáhnout přidáním vhodného mechanismu zajišťujícího, že i v případě, že nastane kolize, bude možné ji ještě vyřešit, aniž by všechny stanice přestaly vysílat a čas, během kterého se již vysílalo, tak byl zcela ztracen. Takovýto mechanismus bývá označován jako CSMA/CR (Collision Resolution) nebo CSMA/CD+AMP (Collision Detection and Arbitration on Message Priority). Mechanismus CSMA/CD+AMP je použit například u sběrnice CAN (viz dále).

Na rozdíl od metod s náhodným přístupem, nedochází u metod s řízeným přístupem ke kolizím a je možné stanovit časový interval, za který příslušná stanice obdrží právo k přenosu. Nejjednodušší je uspořádání master-slave, v němž je pouze jedna řídicí stanice (master), která jako jediná má právo zahájit komunikaci a vyzvat některou z podřízených stanic (slave) k vysílání dat. Toto vyzývání je nejčastěji označováno jako cyklická výzva (cyclic polling). Další metodou je například model označovaný jako producent-distributor-konzument (PDC - Producer-Distributor-Consumer).

Jistým problémem u metod s náhodným přístupem je závislost sítě na řídicí stanici a s tím související zhroucení sítě při jejím výpadku.

K dalším charakteristikám sítí definovaných na úrovni spojové vrstvy je zabezpečení dat. Obecně platí, že v důsledku rušivých vlivů bude vždy určitá část dat přijata chybně. Jistá opatření pro snížení chybovosti lze učinit již na úrovni fyzické vrstvy (symetrické rozhraní, stínění, atd.).

Určitá citlivost na rušení však vznikne vždy a z toho důvodu jsou na rovině spojové vrstvy definovány kódy pro detekci případně opravu chyb. Jsou to:

- paritní kód

- iterační kódy s podélnou a příčnou paritou
- cyklické kódy (např. CRC - Cyclic Redundancy Check)

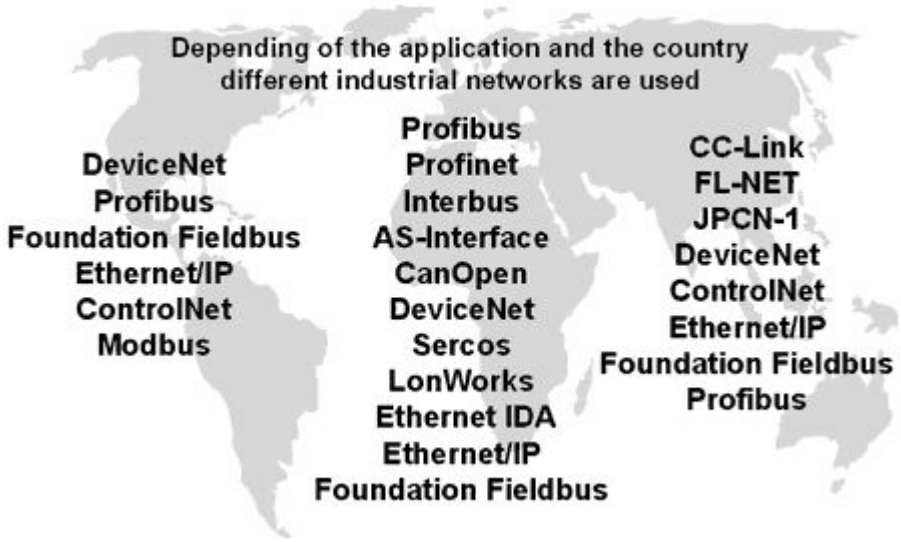
### 2.6.4 Přehled otevřených komunikačních systémů

Průmyslové komunikační systémy označované jako fieldbus jsou svojí flexibilitou vhodné pro nasazení v mnoha aplikacích. Při velké variabilitě aplikací je nutné zajistit potřebné programové vybavení a pro každou aplikaci vlastní SW se standardními knihovnami.

#### *2.6.4.1 Stav a vývoj*

Jak již bylo zmíněno, v současné době existuje velké množství fieldbusů, které se liší způsobem přístupu k přenosovému mediu, přenosovým protokolem, zabezpečením přenosu, aplikační vrstvou atd.

Použití ve světě v závislosti na místě aplikace je znázorněno na obr. 2.6-4.




*Obr. 2.6-4 Přehled používaných typů sběrnic ve světě v závislosti na aplikaci a zemi použití*

#### *2.6.4.2 Členění dle použití*

2.6.4.2.1 Komunikační vrstva FAN: Systembus (Sběrnice pro systémovou úroveň)

- PROFIBUS-FMS (euronorma)
- WorldFIP (euronorma)
- P-NET (euronorma)
- INTERBUS (návrh euronormy)
- Modnet/Modbus
- ARCNET
- FOUNDATION Fieldbus (IEC-Fieldbus norma)

2.6.4.2.2 Komunikační vrstva FAN: Objektově orientovaná systémová úroveň

- PROFIBUS-DP (euronorma)
- PROFIBUS-PA (euronorma)
- DIN-Messbus
- SERCOS (euronorma, IEC-norma)
- BITBUS (IEEE-norma)
- CAN (ISO-norma)
- LON (IEC-návrh normy)

2.6.4.2.3 Komunikační vrstva FAN: Sběrnice pro úroveň snímačů a akčních členů

- AS-Interface (IEC- návrh normy)
- INTERBUS-Loop

- HART
- EIB (euronorma)
- M-Bus (návrh euronormy)
- eBUS

2.6.4.2.4 Komunikační úroveň LAN: Průmyslový Ethernet

- Ethernet Powerlink
- Ethernet/IP

#### *2.6.4.3 Přehledové tabulky*

Vlastnosti jednotlivých sběrnic přehledně znázorňují tabulky v Příloze A-1 a Příloze A-2 na stranách 116 a 117.

## *2.7 Popis vybraných typů průmyslových sběrnic*

### 2.7.1 CAN

#### *2.7.1.1 Vznik, vývoj a současné postavení*

*Controller Area Network* (CAN) je sériový komunikační protokol, který byl původně vyvinut firmou Bosch pro nasazení v automobilech. Vzhledem k tomu, že přední výrobci integrovaných obvodů implementovali podporu protokolu CAN do svých produktů, dochází k stále častějšímu využívání tohoto protokolu i v různých průmyslových aplikacích. Důvodem je především nízká cena, snadné nasazení, spolehlivost, vysoká přenosová rychlost, snadná rozšiřitelnost a dostupnost potřebné součástkové základny.

V současné době má protokol CAN své pevné místo mezi ostatními fieldbusy a je definován normou ISO 11898. Ta popisuje fyzickou vrstvu protokolu a specifikaci CAN 2.0A. Později byla ještě vytvořena specifikace CAN 2.0B, která zavádí dva pojmy - standardní a rozšířený formát zprávy (lišící se v délce identifikátoru zprávy). Tyto dokumenty definují pouze fyzickou a linkovou vrstvu protokolu podle referenčního modelu ISO/OSI. Aplikační vrstva protokolu CAN je definována několika vzájemně nekompatibilními standardy (CAL/CANopen, DeviceNet, CAN Kingdom, Smart Distributed System).

#### *2.7.1.2 Základní vlastnosti protokolu CAN:*

CAN byl navržen tak, aby umožnil provádět distribuované řízení systémů v reálném čase s přenosovou rychlostí do 1Mbit/s a vysokým stupněm zabezpečení přenosu proti chybám. Jedná se o protokol typu *multi-master*, kde každý uzel sběrnice může být *master* a řídit tak chování jiných uzlů. Není tedy nutné řídit celou síť z jednoho "nadřazeného" uzlu, což přináší zjednodušení řízení a zvyšuje spolehlivost (při poruše jednoho uzlu může zbytek sítě pracovat dál). Pro řízení přístupu k médiu je použita sběrnice s náhodným přístupem, která řeší kolize na základě prioritního rozhodování. Po sběrnici probíhá komunikace mezi dvěma uzly pomocí zpráv (datová zpráva a žádost o data), a management sítě (signalizace chyb, pozastavení komunikace) je zajištěn pomocí dvou speciálních zpráv (chybové zprávy a zprávy o přetížení).

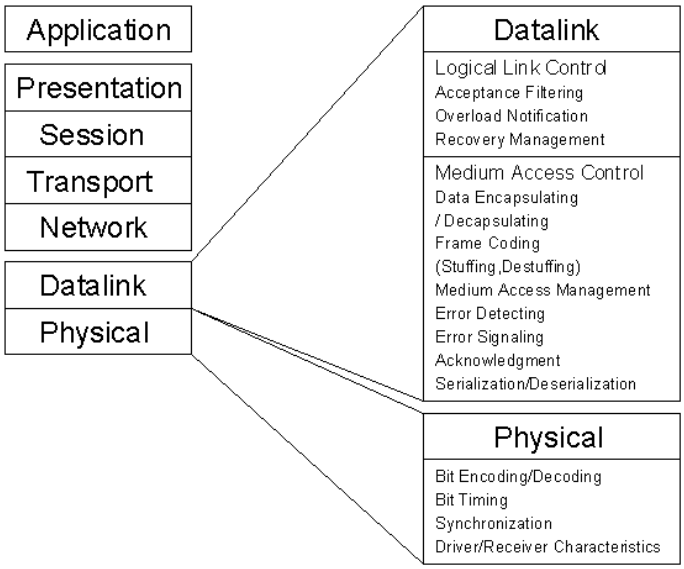
Zprávy vysílané po sběrnici protokolem CAN neobsahují žádnou informaci o cílovém uzlu, kterému jsou určeny, a jsou přijímány všemi ostatními uzly připojenými ke sběrnici. Každá zpráva je uvozena identifikátorem, který udává význam přenášené zprávy a její prioritu, nejvyšší prioritu má zpráva s identifikátorem 0. Protokol CAN zajišťuje, aby zpráva s vyšší prioritou byla v případě kolize dvou zpráv doručena přednostně a dále je možné na základě identifikátoru zajistit, aby uzel přijímal pouze ty zprávy, které se ho týkají (*Acceptance Filtering*).

2.7.1.2.1 CAN - Přehled některých parametrů

| | |
|---|---|
| Přenosová rychlost | 125 kBit/s až 1 Mbit/s |
| Počet uzlů v síti | max. 30 |
| Přenosová rychlost: | odpovídající maximální délka: |
| 1 Mbit/s | 40 m |
| 500 kbit/s | 112 m |
| 300 kbit/s | 200 m |
| 100 kbit/s | 640 m |
| 50 kbit/s | 1340 m |
| 20 kbit/s | 2600 m |
| 10 kbit/s | 5200 m |
| Charakteristická impedance vedení | 120 Ohmů |

*Tab. 2.7-1 Vybrané parametry CAN*

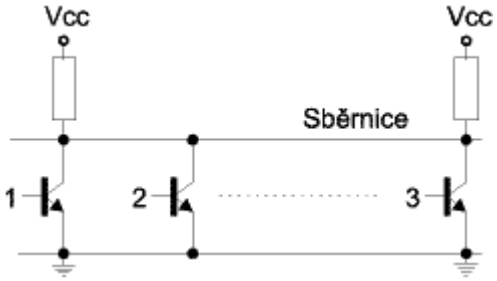
### *2.7.1.3 Analogie vrstev s OSI modelem*




*Obr. 2.7-1 Analogie s OSI modelem*

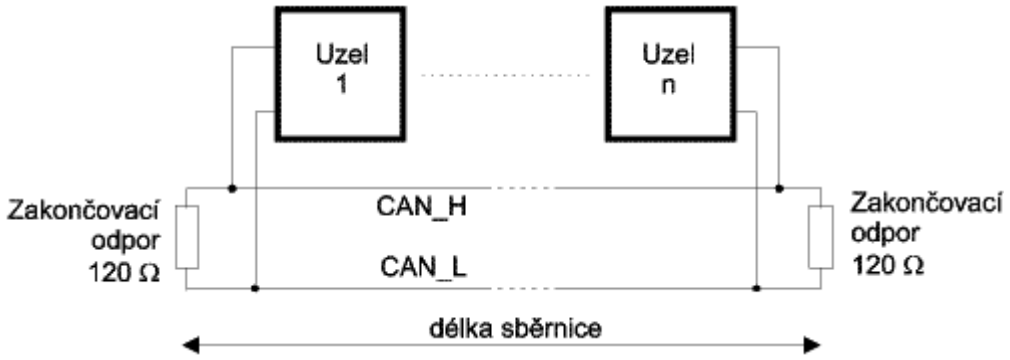
2.7.1.3.1 Fyzická vrstva

Základním požadavkem na fyzické přenosové médium protokolu CAN je, aby realizovalo funkci logického součinu. Standard protokolu CAN definuje dvě vzájemně komplementární hodnoty bitů na sběrnici - *dominant* (logická 0) a *recessive* (logická 1). Jedná se v podstatě o jakýsi zobecnělý ekvivalent logických úrovní, jejichž hodnoty nejsou určeny a skutečná reprezentace záleží na konkrétní realizaci fyzické vrstvy. Pravidla pro stav na sběrnici jsou jednoduchá a jednoznačná. Vysílají-li všechny uzly sběrnice *recessive* bit, pak na sběrnici je úroveň *recessive*. Vysílá-li alespoň jeden uzel *dominant* bit, je na sběrnici úroveň *dominant*. Příkladem může být optické vlákno, kde stavu *dominant* bude odpovídat stav "svítí" a *recessive* stav "nesvítí". Dalším příkladem může být sběrnice buzená hradly s otevřeným kolektorem (obr. 2.7-2), kde stavu *dominant* bude odpovídat logická nula na sběrnici a stavu *recessive* logická jednička. Pak, je-li jeden tranzistor sepnut, je na sběrnici úroveň logické nuly (*dominant*) a nezáleží již na tom, zda je či není sepnutý i nějaký jiný tranzistor. Pokud není sepnut žádný tranzistor, je na sběrnici úroveň logické jedničky (*recessive*).




*Obr. 2.7-2 Příklad realizace fyzické vrstvy protokolu CAN*

Pro realizaci fyzického přenosového média se nejčastěji používá diferenciální sběrnice definovaná podle normy ISO 11898. Tato norma definuje jednak elektrické vlastnosti vysílacího budiče a přijímače a zároveň principy časování, synchronizace a kódování jednotlivých bitů. Sběrnici tvoří dva vodiče (označované CAN_H a CAN_L), kde *dominant* či *recessive* úroveň na sběrnici je definována rozdílovým napětím těchto dvou vodičů. Dle nominálních úrovní uvedených v normě je pro úroveň *recessive* velikost rozdílového napětí $V_{diff}$ = 0V a pro úroveň *dominant* $V_{diff}$ =2V. Pro eliminaci odrazů na vedení je sběrnice na obou koncích přizpůsobena zakončovacími odpory o velikosti 124 Ohmů. Jednotlivá zařízení jsou na sběrnici připojena pomocí konektorů, nejčastěji jsou používány konektory D-SUB.




*Obr. 2.7-3 Principielní struktura sítě CAN podle ISO 11898*

Přesné elektrické charakteristiky sběrnice jsou detailně popsány v normě ISO 11898, v této práci je uvedeno pouhé shrnutí několika nejdůležitějších a pro praktické využití užitečných faktů týkajících se fyzické vrstvy protokolu CAN.

Na sběrnici může být čistě teoreticky připojeno neomezené množství uzlů, avšak s ohledem na zatížení sběrnice a zajištění správných statických i dynamických parametrů sběrnice norma uvádí jako maximum 30 uzlů připojených na sběrnici. Maximální délka sběrnice je pro přenosovou rychlost 1Mbit/s udána normou 40 m. Pro jiné přenosové rychlosti délku sběrnice norma neudává, avšak lze logickým úsudkem dojít k závěru, že pro nižší přenosové frekvence bude maximální délka sběrnice větší. Maximální délky sběrnice pro jinou přenosovou rychlost než 1Mbit/s uvedené v tab. 2.7-1 jsou pouze informativní a závisí na mnoha parametrech (např. typu použitého kabelu).

2.7.1.3.2 Aplikační vrstva

V současné době jsou nejrozšířenější protokoly implementující jednotlivé služby aplikační vrstvy protokoly CAL/CANopen, DeviceNet, CanIO, Smart Distributed System a CAN Kingdom.

#### *2.7.1.4 Další specifické vlastnosti - základní typy zpráv*

Specifikace protokolu CAN definuje čtyři typy zpráv.

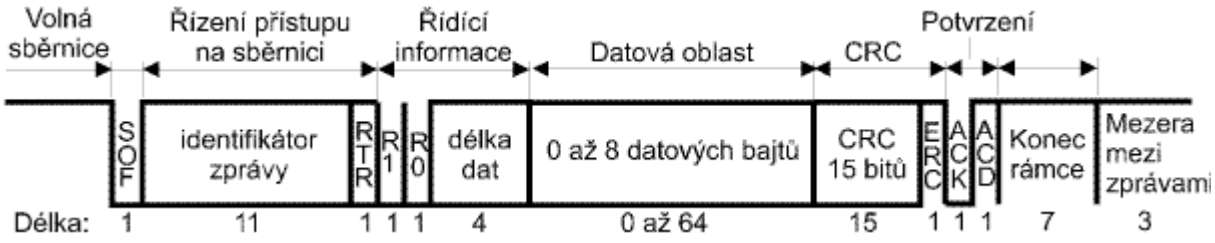
První dvě se týkají datové komunikace po sběrnici. Je to jednak datová zpráva, která představuje základní prvek komunikace uzlů po sběrnici, a dále pak zpráva na vyžádání dat, kdy uzel žádá ostatní účastníky na sběrnici o zaslání požadovaných dat. Datová zpráva umožňuje vyslat na sběrnici 0 až 8 datových bajtů. Pro jednoduché příkazy uzlům (např. příkazy typu vypni/zapni) není nutné přenášet žádné datové bajty (význam příkazu je dán identifikátorem zprávy), což zkracuje dobu potřebnou k přenosu zprávy a zároveň zvětšuje propustnost sběrnice, zvláště pak při silném zatížení. Zpráva na vyžádání dat je vyslána uzlem, který požaduje zaslání určitých dat. Odpovědí na tento požadavek je odeslání požadovaných dat uzlem, který tato data má k dispozici.

Poslední dvě zprávy (chybová zpráva a zpráva o přetížení) slouží k managementu komunikace po sběrnici, konkrétně k signalizaci detekovaných chyb, eliminaci chybných zpráv a vyžádání prodlevy v komunikaci.

2.7.1.4.1 Datová zpráva (Data Frame)

Protokol CAN používá dva typy datových zpráv. První typ je definován specifikací 2.0A a je v literatuře označován jako standardní formát zprávy (*Standard Frame*), zatímco specifikace 2.0B definuje navíc tzv. rozšířený formát zprávy (*Extended Frame*). Jediný podstatný rozdíl mezi oběma formáty je v délce identifikátoru zprávy, která je 11 bitů pro standardní formát a 29 bitů pro rozšířený formát. Oba dva typy zpráv mohou být používány na jedné sběrnici, pokud je použitým řadičem podporován protokol 2.0B.

Vyslání datové zprávy je možné pouze tehdy, je-li sběrnice volná (stav *Bus Free*). Jakmile uzel, který má připravenu zprávu k vyslání, detekuje volnou sběrnici, začíná vysílat. Zda získá přístup na sběrnici či nikoliv, záleží na již popsaném mechanismu řízení přístupu k médiu. Strukturu datové zprávy podle specifikace 2.0A ilustruje obr. 2.7-4.




*Obr. 2.7-4 Datová zpráva podle specifikace CAN 2.0A*

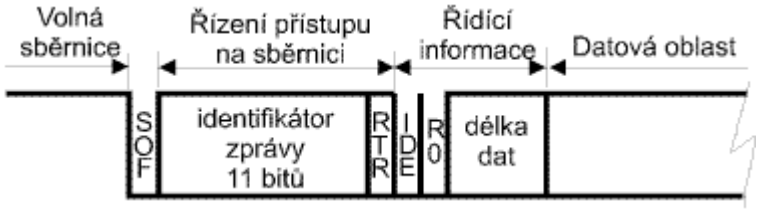
Význam jednotlivých částí datové zprávy podle specifikace 2.0A je následující:

- **Začátek zprávy** (*SOF = Start Of Frame*)- začátek zprávy, 1 bit *dominant*.
- **Řízení přístupu na sběrnici** (*Arbitration Field*) - určení priority zprávy
- **Identifikátor zprávy** - 11 bitů, udává význam přenášené zprávy
- **RTR bit** (*Remote Request*) - 1 bit, příznak udává, zda se jedná o datovou zprávu nebo o žádost o vyslání dat. V datové zprávě musí být tento bit *dominant*, v žádosti o data *recessive*.
- **Řídicí informace** (*Control Field*)
- **R0, R1** - rezervované bity
- **Délka dat** - 4 bity, počet přenášených datových bajtů ve zprávě. Povolené hodnoty jsou 0 až 8.
- **Datová oblast** (*Data Field*) - datové bajty zprávy. Maximálně 8 bajtů je vysláno od MSB
- **CRC** (*CRC Field*) - 16 bitů, zabezpečovací CRC kód
- **CRC kód** - 15 bitů
- **ERC** (*CRC oddělovač*) - 1 bit *recessive*
- **Potvrzení** (*ACK Field*) - 2 bity
- **ACK** (bit potvrzení) - 1 bit
- **ACD** (oddělovač potvrzení) - 1 bit *recessive*

- **Konec zprávy** (*End Of Frame*) - 7 bitů *recessive*
- **Mezera mezi zprávami** (*Interframe Space*) - 3 bity *recessive*, odděluje dvě zprávy

Specifikace CAN 2.0B definuje dva formáty datového zprávy - standardní a rozšířený.

**Standardní zpráva** (*Standard Frame*) je převzata ze specifikace 2.0A, má délku identifikátoru zprávy 11 bitů. Jediným rozdílem je zde využití bitu R1 na indikaci, zda se jedná o rámec standardní nebo rozšířený. Zde se podle CAN 2.0B tento bit označuje IDE (*Identifier Extended*) a je *dominant* pro standardní formát a *recessive* pro rozšířený formát zprávy. Z obr. 2.7-6, který zobrazuje začátek rámce je vidět, že řízení přístupu na sběrnici (priorita zprávy) je dána opět 11-ti bity identifikátoru a hodnotou bitu RTR (*Remote Request*).




*Obr. 2.7-5 Začátek datové zprávy (standardní formát) podle specifikace 2.0B*

**Rozšířený rámec** (*Extended Frame*) používá celkem 29 bitový identifikátor zprávy. Ten je rozdělen do dvou částí o délkách 11 (stejný identifikátor je použit ve standardním formátu) a 18 bitů (viz obr. 2.7-6). Bit RTR (*Remote Request*) je zde nahrazen bitem SRR (*Substitute Remote Request*), který má v rozšířeném formátu vždy hodnotu *recessive*. To zajišťuje, aby při vzájemné kolizi standardního a rozšířeného formátu zprávy na jedné sběrnici se stejným 11-ti bitovým identifikátorem, získal přednost standardní rámec. Bit IDE (*Identifier Extended*) má vždy *recessive* hodnotu. Bit (RTR) udávající, zda se jedná o datovou zprávu nebo žádost o data je přesunut za konec druhé části identifikátoru. Pro řízení přístupu k médiu jsou použity ID (11 bit), SRR, IDE, ID (18 bit), RTR. V tomto pořadí je určena priorita datové zprávy.


*Obr. 2.7-6 Začátek datové zprávy (rozšířený formát) podle specifikace 2.0B*

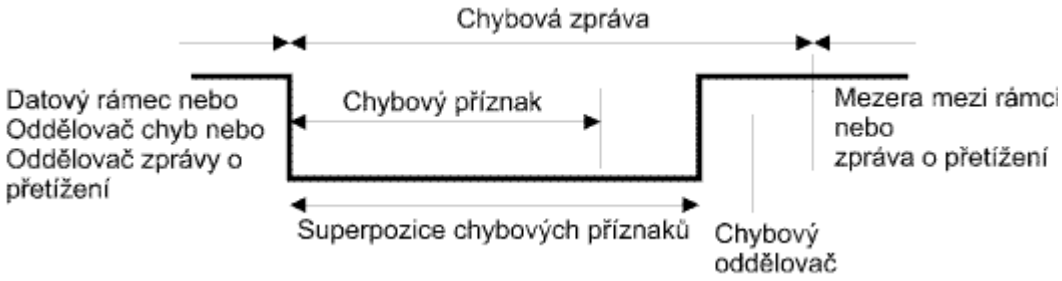
2.7.1.4.2 Žádost o data (Remote Frame)

Formát žádosti o data je podobný jako formát datové zprávy. Pouze je zde RTR bit (pole řízení přístupu na sběrnici) nastaven do úrovně *recessive* a chybí datová oblast. Pokud nějaký uzel žádá o zaslání dat, nastaví takový identifikátor zprávy, jako má datová zpráva, jejíž zaslání požaduje. Tím je zajištěno, že pokud ve stejném okamžiku jeden uzel žádá o zaslání dat a jiný data se stejným identifikátorem vysílá, přednost v přístupu na sběrnici získá uzel vysílající datovou zprávu, neboť úroveň RTR bitu datové zprávy je *dominant* a tudíž má tato zpráva vyšší prioritu.

2.7.1.4.3 Chybová zpráva (Error Frame)

Chybová zpráva slouží k signalizaci chyb na sběrnici CAN. Jakmile libovolný uzel na sběrnici detekuje v přenášené zprávě chybu (chyba bitu, chyba CRC, chyba vkládání bitů, chyba rám-

ce), vygeneruje ihned na sběrnici chybový rámec. Podle toho, v jakém stavu pro hlášení chyb se uzel, který zjistil chybu, právě nachází, generuje na sběrnici buď aktivní (šest bitů *dominant*) nebo pasivní (šest bitů recessive) příznak chyby. Při generování aktivního příznaku chyby je přenášená zpráva poškozena (vzhledem k porušení pravidla na vkládání bitů), a tedy i ostatní uzly začnou vysílat chybové zprávy. Hlášení chyb je pak indikováno superpozicí všech chybových příznaků, které vysílají jednotlivé uzly. Délka tohoto úseku může být minimálně 6 a maximálně 12 bitů.

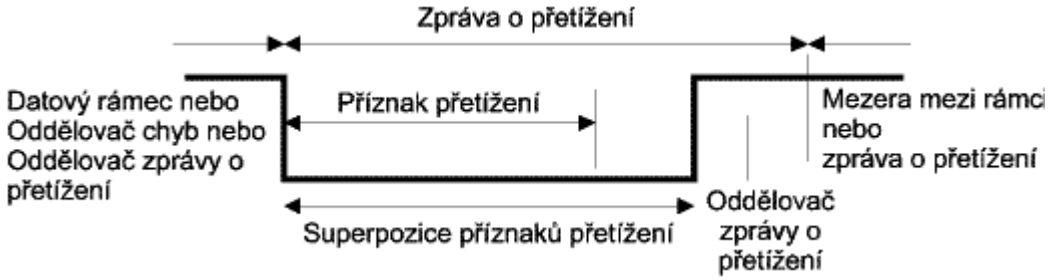



*Obr. 2.7-7 Chybová zpráva protokolu CAN*

Po vyslání chybového příznaku vysílá každá stanice na sběrnici bity *recessive*. Zároveň detekuje stav sběrnice a jakmile najde první bit na sběrnici ve stavu *recessive*, vysílá se dalších sedm bitů *recessive*, které plní funkci oddělovače chyb (ukončení chybové zprávy).

2.7.1.4.4 Zpráva o přetížení (Overload Frame)

Zpráva o přetížení slouží k oddálení vyslání další datové zprávy nebo žádosti o data. Zpravidla tento způsob využívají zařízení, která nejsou schopna kvůli svému vytížení přijímat a zpracovávat další zprávy. Struktura zprávy je podobná zprávě o chybě, ale její vysílání může být zahájeno po konci zprávy (*End of Frame*), oddělovače chyb nebo předcházejícího oddělovače zpráv přetížení.




*Obr. 2.7-8 Zpráva o přetížení*

Zpráva o přetížení je složena z příznaku přetížení (šest bitů *dominant*) a případné superpozice všech příznaků přetížení, pokud jsou generovány více uzly současně. Za příznaky přetížení následuje dalších sedm bitů recessive, které tvoří oddělovač zprávy o přetížení.

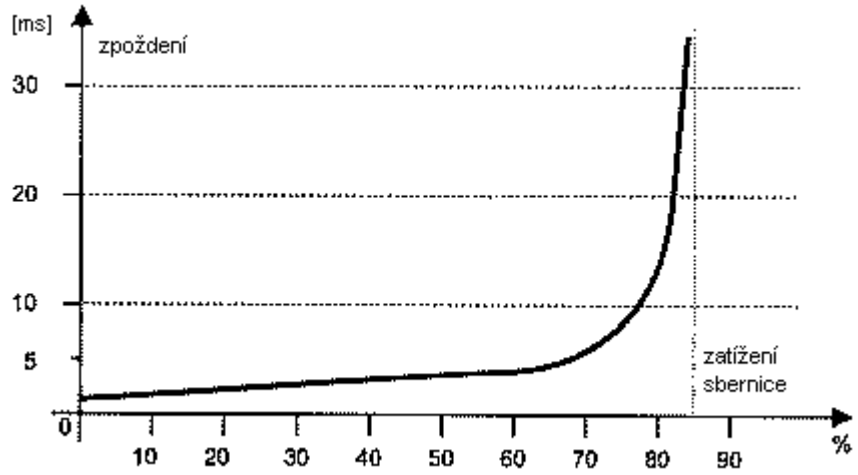
## 2.7.2 Průmyslový Ethernet

### *2.7.2.1 Vznik, vývoj a současné postavení*

Ještě před pár lety by použití Ethernetu v automatizaci pro řízení procesů v reálném čase bylo nemyslitelné. Nevhodnost použití klasického Ethernetu pro časově náročné aplikace, kde je nutná garantovaná doba odezvy, vyplývá již přímo z principu definice protokolu Ethernet.

Poprvé se Ethernet objevil v souvislosti s firmou Xerox, která pod označením Ethernet přivedla na trh lokální síť pro komunikaci až 100 účastníků pomocí koaxiálního kabelu až 1000 m dlouhého. Ethernet jako komunikační protokol byl definován v roce 1984 jako standard

IEEE 802.3 s rozšířením na 1024 účastníků a přenosovou rychlost 10 Mbit/s. Jedna z hlavních výhod je jeho jednoduchost. Přístup na sběrnici je založen na náhodném principu CSMA/CD (popsán již dříve v souvislosti s referenčním modelem OSI). To vede ke zpoždění signálů, které může vyústit až v úplné zablokování přenosu po sběrnici. (viz. obr. 2.7-9)




*Obr. 2.7-9 Závislost zpoždění na využití sběrnice*

Navzdory tomuto problému se Ethernet, především ve spojení s nadřazenými komunikačními protokoly TCP/IP, prosadil jako komunikační standard v kancelářském světe. Zde se prokázal jako spolehlivý, rychlý, finančně relativně nenáročný a pro přenos v kancelářských aplikacích bezrizikový. Právě díky tomuto rozšíření došlo i k pronikání do průmyslových automatizačních systémů ne jako strategická komunikační platforma ale jako praktický prostředek pro spojení průmyslových řídicích systémů s LAN sítěmi.

S postupem doby se ukázalo jako vhodné použití Ethernetu i jako komunikační sběrnice pro přímé řízení průmyslového procesu. Důvodů, proč používat Ethernet v automatizaci je hned několik.

- Jak již bylo zmíněno, Ethernet představuje standard v oblasti kancelářských komunikačních aplikací.
- Masové použití Ethernetu umožňuje nízké ceny jednotlivých komponent a jejich snadnou dostupnost.
- Množství aplikací Ethernetu v současné době zaručuje jeho další vývoj a tím tedy i budoucnost pro sběrnice založené na Ethernetu.
- Možnost relativně jednoduchého připojení na Internet
- V neposlední řadě pak vysoká přenosová rychlost. V současné době je standardem přenosová rychlost 100 Mbit/s a pomalu nastupuje již 1000 Mbit/s („Gigabitový“) Ethernet.

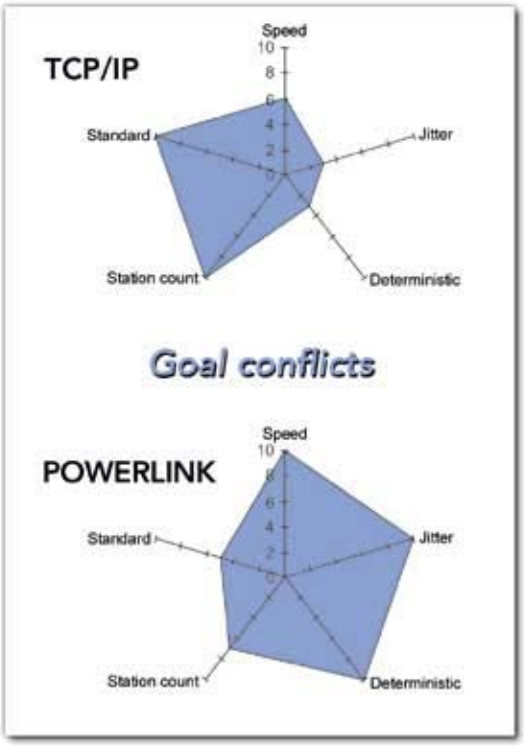
Využití Ethernetu v automatizaci je však třeba rozdělit do dvou z hlediska časové náročnosti podstatně odlišných skupin:

- časově nenáročné aplikace pro přenos velkého objemu dat
- aplikace určené pro řízení v reálném čase

Pro první skupinu, kdy není nutná garantovaná doba odezvy a vysoká spolehlivost, je vyhovující standardní použití Ethernetu s protokoly TCP/IP.

Pro druhou skupinu již však použití Ethernetu s TCP/IP není bez omezení reálného času možné. Zatím není možné realizovat univerzální síť pokrývající celé spektrum od senzorů, programovatelných automatů až po osobní počítače. Důvodem k tomu je použití takzvaných switchů - komponent v síti, které analyzují příchozí data a následně je selektivně posílají dál. Tato činnost v porovnání s pouhými Huby (rozbočovači) představuje dlouhé latentní (reakční) časy a způsobuje kolísání, tedy rozptyl (Jitter).

Základními požadavky pro síť aplikovatelnou v této oblasti automatizace jsou tedy rychlost, schopnost deterministického chování a nízký rozptyl.




*Obr. 2.7-10 Porovnání vlastností komunikace založené na TCP/IP a Powerlinku*

Na obr. 2.7-10 je uvedeno porovnání vlastností klasické komunikace po Ethernetu s TCP/IP protokoly a jednoho z možných řešení problému komunikace v reálném čase - protokolu Powerlink firmy Bernecker&Rainer. Je porovnáno pět parametrů - rychlost (Speed), rozptyl (Jiter), do jaké míry je komunikace deterministická (Deterministic), možný počet připojených stanic (Station Count), a míra standardu (Standard).

Stupnice jsou rozděleny na deset dílků a hodnoty jsou uvedeny poměrně vůči sobě - u rychlosti a počtu stanic hodnota 10 odpovídá nejvyššímu možnému počtu stanic příslušné varianty, u standardu resp. determinismu pak hodnota 10 odpovídá míře standardu resp. míře determinismu. Hodnota 10 rozptylu představuje nejkvalitnější rozptyl (nejmenší hodnotu).

Je patrné, že důsledkem zlepšení vlastností Powerlinku u rychlosti, rozptylu a determinismu je zhoršení standardizace a zmenšení možného počtu připojených stanic.

V současné době dochází k značnému rozvoji použití Ethernetu v průmyslové automatizaci. Každý výrobce nabízí standardně u velké části svých produktů Ethernetové rozhraní. Zatím je toto rozhraní používáno především jako prostředek pro přenos informací o stavu řízeného procesu a Ethernet není využíván jako prostředek pro řízení samotné, i když již i počet těchto aplikací roste a do budoucna bude nepochybně častější a častější.

V části 2.7.2.2 je nejdříve ještě zmíněna problematika bezpečnosti s masivním nástupem komunikace pomocí Ethernetu do automatizace a s tím i nová bezpečnostní rizika v automatizaci známá doposud jen v oblasti internetu.

Další část 2.7.2.3.1 je zaměřena na detailnější popis vybrané komunikační sběrnice - Ethernet Powerlink, která je již schopna práce v reálném čase a je tedy i vhodná jako sběrnice pro přímé řízení průmyslových procesů.

### *2.7.2.2 Ethernet v průmyslovém nasazení a bezpečnostní rizika*

Současná automatizační technika stojí v současné době před úlohou vyvinout nové strategie pro vysoce flexibilní, dostupnou a cenově výhodnou výrobu. Tato úloha pomalu krystalizuje v řešení, které spočívá ve všeobecně průchodném síťovém spojení automatizačních komponent a to od oblasti kanceláří přes řízení produkce až dolů po úroveň rychlého řízení strojů a synchronizace - a to právě s použitím Ethernetu.

S nasazováním Ethernetu v oblasti automatizace se ale naskýtá nejen otázka schopnosti práce v reálném čase. Kvůli použití komunikace založené na webových technologiích, komponent založených na PC a nasazení standardních operačních systémů vznikají při neošetřené komu-

nikaci v rámci sítě a jejích služeb nová nebezpečí a rizika, se kterými nebylo nutné doposud v této oblasti počítat.

V současné době již existují četné provozy, které jsou pomocí Ethernetu spojeny nejen horizontálně v rámci stejné úrovně, ale rovněž vertikálně právě od oblasti kanceláří až po řízení strojů. Napojení programovatelných automatů na Ethernetové sítě umožňující archivaci programů a procesních parametrů není žádnou výjimkou.

V rámci této práce není bohužel prostor pro podrobnější popis tohoto nepochybně důležitého problému, detailnější informace je možné nalézt například v [10].

Přesto je však zajímavé uvést následující dva příklady, které nesouvisí s nepochybně zřejmým nebezpečím typu virové nákazy a útoků hackerů. Nebezpečí totiž často „číhá“ přímo uvnitř jako následek nechtěné chyby, která může mít katastrofální následky.

V konkrétním případě se programátor pomocí údaje IP adresy propojil s automatem stroje výrobní linky, o kterém byl přesvědčen, že stojí přímo před ním. Po modifikaci programu a jeho nahrání do automatu se divil, jak to že se změna neprojevila. Ve stejném okamžiku ale zůstala stát v jiné výrobní hale ze zdánlivě nepochopitelných důvodů celá linka. Kvůli špatnému zadání IP adresy došlo totiž ke spojení s jiným PA, jehož program byl pak modifikován. Chyba v zadání IP adresy je ale lehce představitelná a díky neexistenci určitého ověření totožnosti a autorizace pak může dojít k fatálním následkům.

Z vlastní konkrétní zkušenosti pak mohu uvést příklad komunikace s zařízením B&R Power Panel. Tento kompaktní programovatelný automat s operátorským displejem (takzvané OPLC) je vybaven Ethernetovým rozhraním umožňujícím komunikaci přes protokol TCP/IP. Díky němu lze například programovat toto OPLC z prostředí Automation Studio po rychlém Ethernetovém rozhraní namísto pomalého RS232. Dále je pak možné tímto způsobem komunikovat například s OPC serverem. Kromě jiného pak v rámci OPLC běží FTP server, který umožňuje přístup na kartu Compact Flash, sloužící jako paměťové médium pro uložení programů (použití obdobně jako EEPROM).

Výrobcem standardně nastavená je možnost přístupu z libovolného FTP klienta (bez hesla) přes FTP server na OPLC přímo na celou paměťovou oblast karty Compact Flash. V momentě, kdy je tedy Power Panel napojen na síť v rámci podniku, je možné například pouze z Internet Exploreru zadat adresu ftp://[IP_adresa_PowerPanel] a například smazat systémové soubory!!! Katastrofální důsledky se sice projeví až po příštím (re)startu programovatelného automatu, protože ve spuštěném PA jsou čteny z paměti RAM, ale rozhodně se jedná o nepřípustnou možnost.

Té je možné zabránit rozdělením paměťového prostoru karty na dvě partition – systémovou a uživatelskou. Pokud toto nebylo provedeno, jedná se o značné bezpečnostní riziko.

#### *2.7.2.3 Ethernet Powerlink*

2.7.2.3.1 Základní vlastnosti Ethernetu Powerlink

Ethernet Powerlink splňuje kriteria pro obě dříve uvedené hlavní oblasti použití Ethernetu v automatizaci - přenos velkého objemu časově nekritických dat stejně jako datový přenos v reálném čase. Právě schopnost práce v reálném čase zde hraje důležitou roli.

Možnost práce v 400 µs cyklu předurčuje tuto sběrnici pro vysokorychlostní, cyklickou a plně deterministickou datovou komunikaci. Všechny stanice na sběrnici mohou být buď synchronizovány navzájem, nebo vzhledem k programu běžícím na PLC.

Kromě cyklické komunikace může ještě navíc probíhat na pozadí probíhat acyklická výměna určitých dat, aniž by ovlivňovala časově kritickou cyklickou komunikaci. Hodnota rozptylu je menší než 1 µs.

Powerlink - přehled některých parametrů

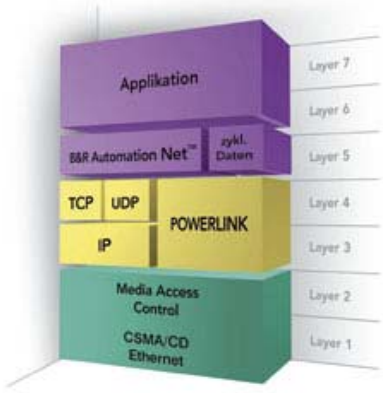
| | |
|---|---|
| Přenos | Standard Fast Ethernet, IEEE 802.3u, 100Base-TX |
| Přenosová rychlost | 100 Mbps |
| Protokol | Slot Communication Network Management (SCNM) |
| Podporované typy komunikace | cyklická, override, acyclická |
| Min. Net Frame | 46 bytes |
| Min. Net Data pro stanici | 1 byte |
| Max. Net Data pro stanici | 1500 bytů |
| Počet stanic v síti Powerlink | Max. 253 |
| Doba cyklu | může být definována |
| Počet stanic při cyklu 400 µs | 8 cyklických + n override stanic (80 bytů net data/stanici) |
| Počet stanic při cyklu 800 µs | 22 cyklických + n override stanic (80 bytů net data/stanici) |
| Možné operační módy stanic | synchronně, asynchronně k cyklu Powerlinku |
| Hardwarový Watchdog | Ano |
| Softwarový Watchdog | Ano |
| Elektrická izolace mezi sběrnicí a I/O | Ano |

2.7.2.3.2 Analogie vrstev s OSI modelem

Na obr. 2.7-11 je znázorněno porovnání jednotlivých vrstev OSI modelu s členěním Ethernetu Powerlinku.

U Powerlinku je jako nejnižší vrstva použita fyzická vrstva Ethernetu spolu s CSMA/CD přístupem a Media Access Control procedurou. Tato vrstva odpovídá fyzické a spojové vrstvě OSI modelu (na obrázku Layer 1 a Layer 2).

Nadřazené vrstvy, u klasického Ethernetu IP a TCP resp. UDP jsou u Powerlinku nahrazeny protokolem Powerlink. To odpovídá síťové a transportní vrstvě OSI modelu (Layer 3 a Layer 4).




*Obr. 2.7-11 Porovnání vrstev modelu OSI a Ethernetu Powerlinku*

Další nadřazená vrstva B&R Automation Net je schopna komunikovat jak s klasickým Ethernetem ve spojení s TCP/IP, tak s protokolem Powerlink. Tato vrstva odpovídá relační vrstvě OSI modelu.

Kromě toho v této vrstvě probíhá ještě cyklická výměna dat s nadřazenou aplikací.

Fyzická vrstva

| | |
|---|---|
| Délka segmentu | Max. 100 m |
| Topologie | Hvězda nebo strom s úrovní dvou Hubů |
| Kabel | Twisted Pair, S/UTP, AWG26 |
| Typ kabelu | Cat. 5 patch cable ('crossover cable'), jeden pro všechny spoje |

2.7.2.3.3 Další specifické vlastnosti

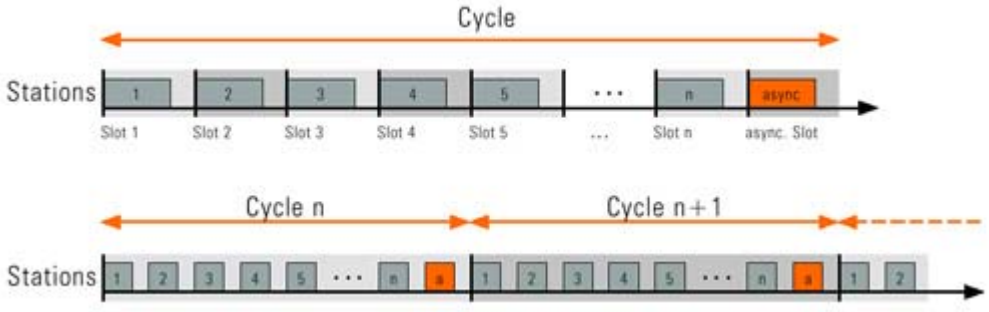
SCNM protokol

Z výše uvedených vlastností Ethernetu je patrné, že u průmyslových sítí založených na Ethernetu je nutno řešit dva problémy.

Prvním je zabránění kolizí. Častým způsobem řešení tohoto problému je jednoduché point-to-point rozdělení, to znamená že pomocí směrovačů (switchů) dojde k segmentaci na jednotlivé kolizní domény. Nevýhodou tohoto přístupu je latentní čas spotřebovaný směrovači a především pak zvýšený rozptyl v síti.

Druhým aspektem k uvážení je výpočetní čas potřebný na zpracování informace v TCP/IP respektive UDP/IP zásobníku (stack). Také tento čas nelze zanedbat a může se pohybovat v řádu několika stovek µs.

Ethernet Powerlink není založen na protokolech TCP/IP a původní zásobník je nahrazen průmyslově vhodnější variantou, která kompletně řídí přenos dat v síti. Tato procedura je nazývána Slot Communication Network Management (SCNM). Na obr. 2.7-12 je stručně znázorněn princip této komunikace. Je zde důsledně využita výhoda vysoké přenosové rychlosti spojená s optimálním vytížením sítě se současně minimálním rozptylem. Každá stanice v síti má časově pevně vymezená vysílací práva. V této době může vysílat data každá stanici připojené k síti. Jedná se tedy o přísnou komunikaci pomocí broadcastingu. Tímto způsobem je zaručeno, že v daném časovém okamžiku může vysílat pouze jeden účastník v síti. Nedochází tedy k žádným kolizím a je možné bezproblémově docílit maximálního vytížení sítě.




*Obr. 2.7-12 Princip Slot Communication Network Management (SCNM)*

Kromě zmíněného synchronního vysílání a přenosu dat od jednotlivých stanic je ještě v rámci každého cyklu vyhrazeno místo pro vysílání asynchronního přenosu. Tímto způsobem je tak možné současně s časově kritickými daty přenášet libovolná data nepodléhající nárokům na reálný čas (například Internetová či Intranetová komunikace).

Otevřenost Powerlinku

Pro budoucnost zmíněného Powerlinku je kromě jeho vlastností neméně významná jeho otevřenost jako standardu, čímž by mohlo dojít k jeho širšímu rozšíření.

# 3 Návrh HW a SW řešení pro dispečerskou úroveň řízení výrobního systému

Návrh řešení linky pro dokončovací operace výroby knihy lze rozdělit do tří částí.

V první části je třeba definovat požadované vlastnosti systému, které pak představují výchozí podmínky zadání pro další návrh konkrétních způsobů řešení vybranými prostředky. Požadavky a navrhované způsoby řešení jsou popsány v kapitole 3.1, kde je uveden požadovaný způsob zadávání dat, jejich struktura a nastíněno řešení tohoto problému. Dále následuje definice požadavků na řídicí systém, programovatelné automaty, komunikační sběrnice. Zařazeny jsou i nároky na identifikaci produktu za účelem detekce chyb a zdůvodnění použití zvoleného identifikačního systému – čárového kódu.

Na základě požadavků definovaných v první části jsem navrhl řešení z hardwarového a softwarového hlediska tak, aby byly splněny dané požadavky. To je předmětem dalších dvou částí - návrh realizace z hardwarového hlediska v kapitole 3.2 a návrh realizace z softwarového hlediska v kapitole 3.3.

## *3.1 Definice požadovaných vlastností systému a prostředky pro jejich splnění*

Základním požadavkem na navrhovanou plně automatickou výrobní linku je kvalita výsledných produktů - knih, které by se neměly lišit od knih vyrobených konvenčním způsobem vyráběných ve velkých nákladech. Tento požadavek vyžaduje splnění určitých podmínek ve dvou základních oblastech.

První oblastí jsou mechanické vlastnosti jednotlivých strojů a schopnosti dosáhnout požadované kvality. Tato oblast není (přímo) předmětem této práce.

Druhou oblastí je automatické řízení strojů. Je zřejmé, že splnění podmínky kvality při plně automatickém provozu s minimálními zásahy člověka vyžaduje sofistikovaný systém řízení.

Proto jako primární způsob použití bude taková struktura zapojení strojů a komunikace mezi nimi, ve které budou jednotlivé stroje podřízené řídicímu systému. Ten bude vyhodnocovat signály ze strojů a senzorů a na jejich základě řídit jednotlivé stroje. Z teoretického hlediska řídicí systém může být buď:

- *decentrální* - jednotlivé stroje spolu komunikují bez přímého řízení některým z nich. V souvislosti s navrhovaným řídicím systémem by bylo možno uplatnit takzvanou *(multi)agentní filosofii*. Na jednotlivých řídicích systémech strojů zařazených ve výrobní lince představuje část softwarového řízení takzvaný „agent“. Ten komunikuje s ostatními agenty a jejich společným cílem je realizace daného výrobního úkolu optimálním způsobem. Mezi hlavní výhody tohoto způsobu realizace obecně patří:
  - řídicí systém je možno realizovat bez nutnosti investice do dalších hardwarových prostředků,
  - pružná reakce na případné poruchy
- *centrální* - řídicí systém představuje jeden předem definovaný počítač (například průmyslové PC). Po sběrnici vyhodnocuje aktuální stav od strojů a senzorů a po sběrnici rovněž určuje, jaké činnosti mají stroje vykonávat.

Mimo jiné z důvodů neexistence podpory agentních systémů u výrobce B&R jsem zvolil variantu centrálního řídicího počítače. Podrobnější rozbor této problematiky je v kapitole 3.1.2.

Tento počítač zároveň představuje jakési "rozhraní s okolním světem", neboli jsou do něj vkládány (ať již elektronicky v datovém souboru určitého formátu či ručním zadáním) data specifikující produkty, které se mají vyrábět.

Avšak stroje (a tedy především jejich navrhovaný řídicí software na programovatelných automatech) musejí být schopny pracovat i samostatně bez nadřazeného řídicího systému (pro případ, kdy stroj není určen pro zařazení do výrobní linky, či pro výpadek nadřazeného řídicího systému). V tomto sekundárním způsobu použití je samozřejmě nutné ovládání obsluhou (především při změně zakázky či jiných parametrů výroby).

### 3.1.1 Vstupní data a sledování procesu

Pro přesnou specifikaci vyráběné knihy je zapotřebí relativně velkého množství údajů, které jsou potřebné pro jednotlivé procesy již zmíněné v části 2.5 a na obr. 2.5-5. Zároveň však pro jednotlivé procesy a jejich data existují určitá pravidla a omezení a soubor těchto dat má v rámci stejného druhu vazby knihy vždy stejnou strukturu. To platí i pro ostatní procesy v polygrafickém průmyslu a proto byl vyvinut (a je dále vyvíjen) takzvaný "Job Definition Format" (JDF), který v sobě obsahuje informace pro všechny procesy v polygrafickém průmyslu.

#### *3.1.1.1 Job Definition Format (JDF)*

Existuje mnoho důvodů pro používání takovéhoto jednotného formátu, který pak přináší řadu výhod.

Důvody vzniku JDF:

V současnosti je možno sledovat především následující trendy v polygrafickém průmyslu :

- kratší doba, po kterou jsou jednotlivé výrobní procesy vykonávány
- více komplexní výrobní procesy

Je zřejmé, že tyto trendy obecně v celém polygrafickém průmyslu jsou podobné s vývojem v konkrétní podoblasti dokončovací výroby knih (přechod na malé náklady) částečně popsaným v kapitolách 2.2, 2.3 a 2.4. Obecně tedy výroba směřuje k plně automatickému výrobnímu postupu (lépe vystihuje anglický termín "workflow"). Ten je poté efektivnější a flexibilnější.

JDF je řešením plně automatického workflow, a to řešením velmi flexibilním a hlavně jednotným. Umožňuje spojit jednotlivé procesy a systémy (často velmi různorodé) a to od okamžiku kdy zákazník odešle objednávku po předání produktu do rukou zákazníka.

Co tedy je konkrétně JDF?

JDF je otevřený průmyslový standard pro polygrafický průmysl spravovaný organizací CIP4, založený na formátu XML. Jméno organizace vystihuje jednotlivé oblasti, pro které je JDF určen. CIP4 je zkratka pro název International Cooperation for the Integration of Processes in Prepress, Press and Post Press (tedy volně přeloženo mezinárodní spolupráce pro integraci procesů v oblastech předtiskové, tiskařské a dokončovací produkce.). Hlavními rysy JDF jsou:

- schopnost nést informace o všech procesech od počáteční fáze výroby produktu až po hotový výrobek
- schopnost překlenout mezeru mezi výrobou a MIS - Management Information Services.
- schopnost překlenout mezeru mezi zákazníkovým pohledem na produkt a výrobním procesem a to pomocí definování pohledu na produkt nezávislého na výrobním procesu stejně tak jako pohledu závislého na výrobě.
- schopnost definovat a sledovat jakýkoli "workflow" definovaný zákazníkem

JDF tedy není počítačovou aplikací ale právě datovým formátem. Vzhledem k zaměření této práce bude dále věnována pozornost jeho části specifikující dokončovací operace. Ještě předtím však bude krátce zaměřena pozornost na důvody, proč je JDF založen na jazyku XML.

#### *3.1.1.2 JDF z pohledu jazyka XML*

JDF je založen na formátu jazyka Extensible Mark up Language (XML) a jeho schéma je konstruováno dle doporučení World Wide Web konsorcia (W3C - World Wide Web Consortium). Důvodů pro použití XML je několik:

- V současné době je XML využíván jako určitá střední, propojovací vrstva pro komunikaci mezi různými integrovanými systémy jako například v systémech ERP. Použití XML pro JDF tak usnadňuje realizaci jednoho z jeho výše zmíněných cílů pro integraci do systémů MIS.
- Jazyk XML není funkční programovací jazyk jako například Java nebo C++, - není tedy možno pomocí něj manipulovat s daty. Jedná se o meta-jazyk, který umožňuje strukturovaně popsat informace (jejich strukturu, vztahy mezi nimi a do určité míry i účel použití). Důležité je pak, že je zde oddělen obsah, struktura a grafická forma prezentace. Jazyk XML je nezávislý na aplikaci, platformě a mediu. Důležitá je rovněž jeho podpora v mnoha programovacích jazycích a databázích.

Hlubší popis není cílem této práce, přesto je však účelné zmínit alespoň některé jeho vlastnosti, důležité pro pochopení navrženého řešení ukládání vstupních dat s použitím JDF/XML.

Základy XML

Podobně jako ostatní jazyky ML (značkovací), i XML používá k strukturovanému zápisu takzvané "tagy".

```
<TAG>obsah</TAG>
```

Je možné použít i takzvaný prázdný tag, syntaxe je ale odlišná než například u HTML

```
<TAG/>
```

Tagy je definován element (nazývaný rovněž uzel), obsah mezi tagy je obsah elementu. Základní myšlenkou strukturovaného zápisu je to, že elementy mohou být do sebe vnořené. Vnořené elementy - pod-elementy jsou nazývány "děti" (children) Nadřazené elementy pak "rodiče" (parents).

Každý XML dokument musí mít následující strukturu.

a) na prvním řádku musí obsahovat deklaraci XML dokumentu
```
<?xml version="1.0"?>
```
b) dále musí následovat takzvaný "kořenový" element (root). Každý XML dokument musí obsahovat *právě jeden* root element.
c) vnořené elementy musí obsahovat počáteční a koncové tagy a musí být správně do sebe vnořené

Struktura XML dokumentu pak tedy může vypadat následujícím způsobem:

```
<?xml version="1.0"?>
<root>
      <element>
            <pod-element>
                  obsah prvního sub-elementu
            </pod-element>
            <pod-element>
                  obsah druhého sub-elementu
            </pod-element>
      <element>
</root>
```

- XML elementy dále mohou obsahovat takzvané "atributy". Syntaxe je následující:

```
<element jmeno_atributu = "hodnota_attributu">....</element>
```

Pokud jsou dodrženy výše zmíněná pravidla a) až c) je dokument takzvaně *"správně formován"* (anglicky *"well-formed"*)

JDF a jeho XML schéma

Jak lze v této souvislosti chápat pojem "schéma"? Pod tímto pojmem je možno si představit "schematickou reprezentaci, náčrt či model". Určitá pravidla, která tedy popisují abstraktní strukturu jisté sady dat je možno nazvat schématem.

XML-schéma je tedy dokument, který popisuje platný formát dat ve struktuře XML. Tato definice určuje, jaké elementy jsou (ale také jaké nejsou) povoleny na jakém místě, jaké atributy mohou být pro které elementy, počet výskytů elementů a tak dále.

Hlavním přínosem XML schématu je v kontrole XML dokumentu a to s pomocí specielního programu nazývaného v angličtině "parser" (to parse = provést rozbor) . "Parser" nejdříve zkontroluje zda dokument je „well-formed“. Pokud ano, přečte pravidla obsažená v XML-schématu a rozhodne, zda kontrolovaný dokument odpovídá těmto pravidlům. V kladném případě pak nahlásí, že dokument je takzvaně *"platný"* (anglicky *"valid"*).

Aplikace těchto pravidel se uplatňuje nejen při kontrole platnosti XML-dokumentu. Sofistikovanější XML-editory umožňují již při návrhu (psaní) XML dokumentu respektovat pravidla daného schématu a nabízejí dle kontextu, které elementy či atributy jsou na daném místě povoleny.

Organizace CIP4 vyvinula a nadále zlepšuje JDF schéma a spolu s ním i specifikaci formátu JDF, která vysvětluje strukturu JDF dokumentů, které samozřejmě musí odpovídat schématu. Existuje tak jednotný návod pro strukturování dat v polygrafickém průmyslu.

#### *3.1.1.3 Dokončovací operace v JDF*

Na základě definice JDF jsem vytvořil specifikace několika základních druhů vazby knihy založené na JDF a to jak pro měkkou vazbu tak i pro tuhou vazbu.

Pro první fázi vývoje linky, která je tématem této práce, je uvažována pouze výroba měkké vazby a to konkrétně lepené měkké vazby.

Pro realizaci tohoto typu knihy jsou zapotřebí následující procesy uvedené v tab. 3.1-1 (také viz část 2.5). Procesy skládání a snášení jsou sice v tabulce a dále v navrženém JDF souboru uvedeny, v konfiguraci linky první fáze vývoje ale nejsou zahrnuty. Procesy uvedené v závorce nejsou bezpodmínečně nutné.

| *česky* | *anglicky* | *německy* |
|---|---|---|
| *skládání* | *Folding* | *Falzen* |
| *snášení* | *Gathering* | *Zusammentragen* |
| opracování hřbetu | Spine Preparation | Buchrückenbearbeitung |
| klížení hřbetu a stran | Perfect Binding | Rücken/Seiten beleimen |
| (lepení gázu) | (Spine Taping) | (Fälzel anlegen) |
| (lepení předsádky) | (End Sheet Gluing) | (Vorsatz ankleben) |
| přilepení obalu | Cover Application | Umschlag anpressen |
| řezání na konečný formát | Trimming | Schneiden auf Endformat |
| balení knih | Stacking and Packaging | Stapeln und Verpacken |

*Tab. 3.1-1 Procesy při výrobě lepené měkké vazby (Softcover)*

Následuje ukázka mnou navrženého a použitého JDF souboru (méně zajímavé části vynechány, celá verze viz 7.2.2 na straně 105).

```
<?xml version="1.0" encoding="UTF-8"?>
<JDF xmlns="http://www.CIP4.org/JDFSchema_1_1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" Type="ProcessGroup" ID="JSoftcover" Status="Cleanup"
xsi:schemaLocation="http://www.CIP4.org/JDFSchema_1_1
E:\MOJE\Diplomka\JDF\JDFSchema_1_1\JDF_1_1.xsd" Version="1.1">
       <NodeInfo DueLevel="Trivial" Start="2002-08-11T12:00:00+02:00" FirstStart="2002-08-
11T11:00:00+02:00" LastStart="2002-08-11T12:30:00+02:00" End="2002-08-11T14:00:00+02:00"
FirstEnd="2002-09-02T13:00:00+02:00" LastEnd="2002-08-11T15:20:00+02:00"
TotalDuration="P00Y00M00DT02H00M" SetupDuration="P00Y00M00DT00H15M"
CleanupDuration="P00Y00M00DT00H30M"/>
       <ResourcePool>
              <FoldingParams ID="Par1" DescriptionType="FoldCatalog" Status="Available"
FoldCatalog="F32-2" Class="Parameter"/>
              <GatheringParams ID="Par2" Status="Available" Class="Parameter">
                     <Disjointing ID="asd" Number="10" Offset="10 10" OffsetAmount="10"
OffsetDirection="Straight"/>
              </GatheringParams>
              <EndSheetGluingParams ID="Par41" Status="Available" Class="Parameter">
                     <EndSheet ID="inp161" Offset="0.000 0.000" Side="Front">
                            <GlueLine ID="inp161GL" StartPosition="0.000 0.000"
WorkingPath="0.000 918.425" AreaGlue="false" GluingPattern="1 0" GlueLineWidth="28.346"
GlueType="ColdGlue" MeltingTemperature="60" GlueBrand="AAAx"/>
                     </EndSheet>
```

```
                        <EndSheet ID="inp162" Offset="0.000 0.000" Side="Back">
                                <GlueLine ID="inp162GL" StartPosition="0.000 0.000"
WorkingPath="0.000 918.425" AreaGlue="false" GluingPattern="1 0" GlueLineWidth="28.346"
GlueType="Hotmelt" MeltingTemperature="60" GlueBrand="BBBx"/>
                        </EndSheet>
                </EndSheetGluingParams>
                <SpinePreparationParams ID="Par3" Status="Available" MillingDepth="5.669"
Class="Parameter" NotchingDistance="0.000" NotchingDepth="0.000"/>
                <SpineTapingParams ID="Par5" Status="Available" Class="Parameter"
HorizontalExcess="14.173" StripLength="918.425" TopExcess="0.000" StripBrand="BrandXYZ"
StripColor="Blue" StripMaterial="Paper">
                        <GlueApplicationRef rRef="GASpineTapingSpine"/>
                        <GlueApplicationRef rRef="GASpineTapingFront"/>
                        <GlueApplicationRef rRef="GASpineTapingBack"/>
                </SpineTapingParams>
                <CoverApplicationParams ID="Par6" CoverOffset="0.000 0.000" Status="Available"
Class="Parameter">
                        <GlueApplicationRef rRef="GACoverSpine"/>
                        <GlueApplicationRef rRef="GACoverFront"/>
                        <GlueApplicationRef rRef="GACoverBack"/>
                </CoverApplicationParams>
                <TrimmingParams ID="Par7" Status="Available" Class="Parameter"
TrimmingType="Detailed" Width="595.276" Height="850.394" TrimmingOffset="39.685"/>
                <StackingParams ID="Par8" Class="Parameter" Status="Available" StandardAmount="50"
MaxAmount="60" MinAmount="15" MaxWeight="15000.000" Compensate="true" Offset="false"
layerAmount="50"/>
...
                <Component ID="Res12" Class="Quantity" Status="Available"
ComponentType="PartialProduct" IsWaste="false" ProductType="unknown" Dimensions="651.968 918.425
170.079"/>
                <Component ID="Res122" Class="Quantity" Status="Available" ComponentType="Sheet"
IsWaste="false" ProductType="unknown" Dimensions="651.968 918.425 2.835"/>
                <Component ID="Res123" Class="Quantity" Status="Available" ComponentType="Sheet"
IsWaste="false" ProductType="unknown" Dimensions="651.968 918.425 2.835"/>
...
                <Component ID="Res152" Class="Quantity" Status="Available" ComponentType="Sheet"
IsWaste="false" ProductType="Cover" Dimensions="651.968 918.425 2.835"/>
                <Component ID="Res16" Class="Quantity" Status="Available" ComponentType="Block"
IsWaste="false" ProductType="unknown" Dimensions="654.803 918.425 181.419"/>
                <Component ID="Res17" Class="Quantity" Status="Available"
ComponentType="FinalProduct" IsWaste="false" ProductType="Book" Dimensions="595.276 850.394
181.419"/>
                <Component ID="Res18" Class="Quantity" Status="Available"
ComponentType="PartialProduct" IsWaste="false" ProductType="unknown" Dimensions="595.276 850.394
9070.95">
...
                </Component>
                <GlueLine ID="GLSpineTapingSpine" Status="Available" Class="Parameter"
WorkingPath="0 918.425" StartPosition="0 0" AreaGlue="false" GluingPattern="1 0"
GlueLineWidth="28.346" GlueType="Hotmelt" GlueBrand="BBBx" MeltingTemperature="60"/>
...
        </ResourcePool>
...
        <JDF ID="J12" Status="Waiting" Type="Gathering" JobID="SRSSoftcover">
                <ResourceLinkPool>
                        <GatheringParamsLink Usage="Input" rRef="Par2"/>
                        <ComponentLink Usage="Input" rRef="Res111" Amount="1000"/>
                        <ComponentLink Usage="Input" rRef="Res112" Amount="1000"/>
                        <ComponentLink Usage="Output" rRef="Res12" Amount="1000"/>
                </ResourceLinkPool>
        </JDF>
        <JDF ID="J13" Status="Waiting" Type="EndSheetGluing" JobID="SRSSoftcover">
                <ResourceLinkPool>
                        <EndSheetGluingParamsLink Usage="Input" rRef="Par41"/>
                        <ComponentLink Usage="Input" rRef="Res12" Amount="1000"/>
                        <ComponentLink Usage="Input" rRef="Res122" Amount="1000"/>
                        <ComponentLink Usage="Input" rRef="Res123" Amount="1000"/>
                        <ComponentLink Usage="Output" rRef="Res13" Amount="1000"/>
                </ResourceLinkPool>
        </JDF>
        <JDF ID="J14" Status="Waiting" Type="SpinePreparation" JobID="SRSSoftcover">
                <ResourceLinkPool>
                        <SpinePreparationParamsLink Usage="Input" rRef="Par3"/>
                        <ComponentLink Usage="Input" rRef="Res13" Amount="1000"/>
                        <ComponentLink Usage="Output" rRef="Res14" Amount="1000"/>
                </ResourceLinkPool>
```

```
        </JDF>
        <JDF ID="J15" Status="Waiting" Type="SpineTaping" JobID="SRSSoftcover">
                <ResourceLinkPool>
                        <SpineTapingParamsLink Usage="Input" rRef="Par5"/>
                        <ComponentLink Usage="Input" rRef="Res14" Amount="1000"/>
                        <ComponentLink Usage="Output" rRef="Res15" Amount="1000"/>
                </ResourceLinkPool>
        </JDF>
        <JDF ID="J16" Status="Waiting" Type="CoverApplication" JobID="SRSSoftcover">
                <ResourceLinkPool>
                        <CoverApplicationParamsLink Usage="Input" rRef="Par6"/>
                        <ComponentLink Usage="Input" rRef="Res15" Amount="1000"/>
                        <ComponentLink Usage="Input" rRef="Res152" Amount="1000"/>
                        <ComponentLink Usage="Output" rRef="Res16" Amount="1000"/>
                </ResourceLinkPool>
        </JDF>
        <JDF ID="J17" Status="Waiting" Type="Trimming" JobID="SRSSoftcover">
                <ResourceLinkPool>
                        <TrimmingParamsLink Usage="Input" rRef="Par7"/>
                        <ComponentLink Usage="Input" rRef="Res16" Amount="1000"/>
                        <ComponentLink Usage="Output" rRef="Res17" Amount="1000"/>
                </ResourceLinkPool>
        </JDF>
        <JDF ID="J18" Status="Waiting" Type="Stacking" JobID="SRSSoftcover" JobPartID="Stack"
Activation="Active">
                <ResourceLinkPool>
                        <StackingParamsLink Usage="Input" rRef="Par8"/>
                        <ComponentLink Usage="Input" rRef="Res17" Amount="1000"/>
                        <ComponentLink Usage="Output" rRef="Res18" Amount="1000"/>
                </ResourceLinkPool>
        </JDF>
        <CustomerInfo CustomerJobName="Softcover" CustomerID="SRS" DescriptiveName="Softcover1">
                <Contact ContactTypes="Customer">
                        <Company OrganizationName="SRS"/>
                </Contact>
        </CustomerInfo>
</JDF>
```

#### *3.1.1.4 Webová aplikace pro práci s JDF soubory*

Jak již bylo uvedeno v úvodu, pro efektivní provoz výrobní linky v knihárně, je důležitá mimo jiné rychlost zadávání zakázek. Především s výhledem do budoucna je tedy vhodná existence určitého elektronického rozhraní mezi zadavatelem zakázky a průmyslovou knihárnou, tedy výrobní linkou, které by umožnilo vytvoření elektronické specifikaci knihy, založené na JDF. Samozřejmým požadavkem pak je možnost přístupu k tomuto rozhraní přes internet/intranet.

Firma SRE mne pověřila vytvořením tohoto rozhraní. Jejím požadavkem bylo, aby specifikace knihy byla samozřejmě založena na standardu JDF a zadávána pomocí standardního webového prohlížeče.

Tyto dva požadavky jsem doplnil o další nutné vlastnosti a na jejich základě jsem navrhl a vytvořil funkční webovou aplikaci, umožňující specifikaci zakázky (knihy) a její následné odeslání k výrobě díky napojení této aplikace na řídicí systém. Dále je uvedena má rozšířená definice na tuto webovou aplikaci a rovněž výběr systému umožňujícího jejich řešení.

##### 3.1.1.4.1 Požadavky kladené na webovou aplikaci

Webová aplikace pro specifikaci zakázky (knihy) musí splňovat následující podmínky

1) Zakázku (její specifikaci) musí být možné zadat pomocí standardního webového prohlížeče.
2) Parametry dokončovací výroby knihy musí být ukládány/načítány z souboru do/z formátu JDF
3) Webová aplikace musí umožňovat přístup více uživatelům

4) Specifikované zakázky musí být možné odeslat do řídicího systému linky, který je pak vyrobí
5) Splnění požadavků bodů 1) až 4) musí být uskutečněno takovým způsobem, aby náklady na dodatečné hardwarové či softwarové komponenty byly minimální.

Tyto body představují základní nároky, které z hlediska řešeného systému byly kladeny na rozhraní pro zadávání dat. V průběhu řešení jsem pak upřesnil a navrhl dodatečné vlastnosti webového rozhraní, rovněž jako postup dalšího zpracování zadaných dat (podrobněji popsáno v kapitole 3.3.1)

Pro první fázi realizace výrobního systému je webová aplikace určena pro komunikaci buď interně v rámci pracovníků knihárny, používající výrobní linku, nebo pro komunikaci s zákazníkem knihárny – nakladatelem, pro usnadnění specifikace knihy.

Výhledově v další fázi bude rozšířená webová aplikace umožňovat specifikaci knihy přímo koncovým zákazníkem –uživatelem služby Print On Demand (viz kapitola 2.4.2).

3.1.1.4.2 Výběr systému umožňujícího řešení daných požadavků

Při řešení daných požadavků jsem vycházel z následujícího předpokladu: zmíněné požadavky vyžadují takový systém, který bude schopen:

- ukládat/načítat data do/z formátu JDF (a tedy bude schopen práce s formátem XML)
- a současně dynamicky generovat soubory ve formátu HTML a umožní tyto stránky procházet z připojených počítačů (pomocí webového prohlížeče).
- pracovat s databází pro autorizaci a správu uživatelů

V současné době existuje mnoho způsobů, jakými programovými prostředky splnit dané zadání. Po rešerši, které systémy jsou schopny provést zmíněné předpokládané operace a zároveň splňují požadavky uvedené v 3.1.1.4.1, jsem vybral následující konfigurace, které dle mého názoru jsou v této souvislosti nejvýhodnější jak z hlediska požadavků tak i programové realizace a i uplatnitelnosti na různých platformách.

- Webový server Apache v kombinaci s programovacím jazykem PHP
- Řešení založené na jazyku Java

JAVA

Realizace v programovacím jazyku JAVA je v zásadě možná dvěma způsoby:

- pomocí systému Java Servlet Pages
- pomocí JAVA appletu

Stručně a zjednodušeně je možno popsat oba způsoby následovně:

Systému Java Servlet Pages jak již název sám vypovídá, pracuje takovým způsobem, že server na kterém běží program zpracovávající kód v jazyce JAVA generuje HTML stránky a provádí příslušné operace.

JAVA applet se načte současně s HTML do prohlížeče a vykoná se daný kód appletu. Výhodou by v tomto případě byla možnost i práce offline – možnost zadání dat do souboru JDF bez nutnosti připojení k serveru (řídicímu počítači). Data by pak mohla být odeslána později.

Apache a PHP (+ MySQL)

Produkty Apache a PHP patří do skupiny Open Source a jsou tedy výhodné z hlediska nákladů na software.

Podstatnou výhodou PHP je relativně jednoduché psaní skriptů a velké množství různých tutoriálů a zdrojových kódů různých řešených problémů dostupných na internetu.¨

Z mnoha důvodů, především kvůli větší jednoduchosti programové realizace v jazyce PHP a v současnosti i jeho dostupnosti přímo v některých řídicích a vizualizačních programech, jsem

se rozhodl pro řešení založené na skriptovacím jazyku PHP (například spolu s webovým serverem Apache a databází MySQL) pro realizaci webové aplikace.

### 3.1.2 Řídicí systém

Z důvodů zmíněných v úvodu části 3.1 jsem zvolil pro řídicí systém variantu centrálního nadřazeného řídicího počítače, který na základě zadaných dat pro jednotlivé zakázky řídí celý proces dokončovacích operací (tedy řídí programovatelné automaty jednotlivých strojů ve dokončovací lince).

S ohledem na dostupné prostředky existuje několik možných variant hardwarového a softwarového uspořádání řídicího systému (Samozřejmým požadavkem pro varianty 1 i 2 je dostupnost sofistikovaného vizualizačního softwaru.):

Varianta 1 – samostatné IPC

Vlastní řídicí systém je realizován samostatným průmyslovým PC (IPC).

Pod běžným operačním systémem (například některé řady Windows ale teoreticky i Linux) běží jak aplikace pro vizualizaci a řízení procesu operátorem, tak i aplikace umožňující komunikaci mezi řídicí a vizualizační aplikací a jednotlivými programovatelnými automaty linky a to přes sběrnici a protokol TCP/IP

*Varianta 1a*

V rámci stejného hardwaru a na pozadí téhož operačního systému je rovněž spuštěn webový server Apache s modulem PHP a databází MySQL.

*Varianta 1b*

Webový server Apache s modulem PHP a databází MySQL je spuštěn na jiném hardwaru. Komunikace je realizována přes standardní síťové služby. Tato varianta, ač finančně náročnější v důsledku dalších investic do hardwar, má výhodu v oddělení webové aplikace od vlastního řídicího systému. Kromě větší bezpečnosti z hlediska přístupu z internetu toto řešení umožňuje vyšší výkon obou subsystémů. Volba operačního systému pro webový subsystém pak není vázána na omezení dostupných softwarových produktů pro komunikaci řídicí systém – programovatelné automaty strojů a může být pak nasazen například operační systém Linux.

Varianta 2 – IPC v rámci stroje linky

Řídicí systém je hardwarově realizován v rámci řídicího automatu jednoho ze strojů výrobní linky. Tím musí být v tomto případě IPC. Paralelně zde běží jak časově kritické řízení stroje (programovatelným automatem) tak operační systém nadřazeného řídicího systému s aplikacemi popsanými ve Variantě 1.

Existují v zásadě dva způsoby – dvě varianty – jak realizovat paralelní běh programů řízení stroje a řídicího systému

*Varianta 2-I*

PA je realizován jako Soft-PLC, tedy softwarově pod stejným operačním systémem jako řídicí systém linky. Operační systém musí ale být uzpůsoben do té míry, že umožňuje zaručené zpracování časově kritického řízení programovatelného automatu, tedy s přiřazením priorit

Rovněž zde je myslitelné stejně jako ve Variantě 1 rozlišit způsob použití webového modulu a a b, ale v tomto případě je dle mého názoru nutné oddělení ve smyslu varianty b.

*Varianta 2-II*

PA je realizován jako přídavná karta s vlastním napájením zasunutá do příslušného slotu IPC. U této varianty je také možné uvažovat obě možnosti a a b uvedené ve Variantě 1.

Varianta 3 - minimální

Především z ekonomických důvodů (s ohledem na finanční možnosti zákazníka kupujícího řídicí systém – knihárny) je možné ještě uvažovat určitou minimální variantu řídicího systému, jehož funkčnost se více méně omezí na centrální zadávání dat pro všechny stroje zapojené v lince (jejíž konfigurace nemusí zahrnovat všechny stroje pro všechny operace výroby knihy) a zjednodušenou analýzu chyb.

Z důvodů již dříve popsané nelehké hospodářské situace v oboru vazby knih si totiž podniky nemohou dovolit značné investice do nových kompletních výrobních linek.

Proto je vhodné nabídnout i takový systém, který umožní knihárně efektivnější zpracování zakázek se sníženými nároky na obsluhu, ale nebude představovat žádné další finanční zatížení v podobě investic do hardwaru a softwarových licencí (které se nutně promítají v ceně prodávaného systému).

*Varianta 3 – minimální může tedy být charakterizována následovně:*

Data pro výrobní linku v její konkrétní konfiguraci mohou být zadávána z jednoho předem určeného stroje. Každému stroji linky jsou pak po sběrnici poslány jeho relevantní data. Smysl tohoto spočívá v odstranění redundantního zadávání stejných údajů v případě jednotlivého zadávání dat strojům.

V této souvislosti je zajímavá myšlenka zjednodušení způsobu zadávání dat (zakázky) například pomocí čárového kódu. Zákazníkovi je možné nabídnout softwarovou aplikaci založenou na vytvořené popisované webové aplikaci určenou k instalaci na jeho stávajícím počítači, například v kanceláři mistra. Webová aplikace pro specifikaci parametrů knihy, založená na JDF souborech, by pak byla rozšířena o následující moduly:

*Modul tisku čárových kódů*

Modul umožňující vytištění jednotlivých dat potřebných k výrobě knihy obsažených v JDF souboru ve formě zakázkového listu (listů) obsahující předem definovanou strukturu čárových kódů. Ty je pak možno snímačem postupně zadat do předem určeného stroje umožňující centrální zadávání dat. Tímto způsobem se značně urychlí zadávání dat.

*FTP Modul*

V případě síťového propojení PC na kterém je nainstalována webová aplikace a stroje určeného k centrálnímu zadávání dat je možné doplnit modul, který přes protokol FTP uloží do paměti stroje soubor s parametry specifikujícími knihu (zakázku) nazvaný dle čísla zakázky. Zároveň vytiskne zakázkový list, který obsahuje čárový kód s unikátním číslem zakázky, které odpovídá názvu souboru. Pro zadání dat je pak nutné pouze zvolit načtení souboru a zadat pomocí snímače číslo zakázky. V době vypracování této práce nebylo v automatech B&R možné načítat přímo soubory ve formátu XML (a tedy i JDF), ale bylo možné načítat soubory ve formátu CSV. Proto v případě předem definované struktury souboru CSV tomuto způsobu zadávání dat nic nebrání

Tuto variantu jsem navrhl, ale dále jsem ji neřešil (a nebyla zatím řešena ani nikým jiným). V této diplomové práci není rovněž dále řešena.

### 3.1.3 PLC

Programování jednotlivých PLC není hlavním předmětem této práce a proto zde nebude podrobně popsáno celé řešení řízení procesů vykonávaných jednotlivými stroji.

Podstatné je, aby byl předem definovaný způsob komunikace mezi jednotlivými podřízenými PLC a řídicím systémem, což je jeden z hlavních problémů řešených v této práci.

V rámci této práce jsem jednak definoval tento způsob komunikace, ale také realizoval tuto komunikaci programově. A to jak na straně řídicího systému (viz část 4.2), tak na straně programovatelných automatů (viz část 4.3).

### 3.1.4 Komunikační sběrnice

Pro úspěšnou realizaci požadavků na řízení celé výrobní linky je zapotřebí spojit jednotlivé podsystémy komunikační sběrnicí s odpovídajícími parametry jako je přenosová rychlost, odolnost proti rušení a schopnost práce danou časovou odezvou a to vše s ohledem na budoucí možné zvýšení objemu přenášených dat v důsledku vylepšení za účelem robustnějšího systému vzhledem k chybám a systému s automatickou detekcí chyb.

Všechny tyto požadavky v podstatě splňují sběrnice zmíněné v kapitole 2.7. Hlavním kritériem je časová odezva. Vzhledem k tomu, že koncepčně řídicí systém musí sice řídit jednotlivé stroje, ale ne přímo časově kritické úlohy související s technologickým procesem, které vykonávají programovatelné automaty jednotlivých strojů, je dostačující použití Ethernetového rozhraní mezi programovatelnými automaty a řídicím systémem.

Sběrnice CAN a Xlink přicházejí v úvahu pro „místní“ spojení mezi senzory a programovatelnými automaty.

### 3.1.5 Identifikace produktu - systém čárového kódu

Identifikace produktu je nezbytná pro zabezpečení detekce případných chyb a jejich nápravu. Z různých možných způsobů, principielně odlišných, je v podstatě nejvýhodnější pro danou výrobní linku systém čárového kódu. Vzhledem k existenci digitálního tiskového stroje předřazenému dokončovací výrobní lince není problém natisknout čárové kódy na ty části knihy, které budou beztak odřezány při operaci na řezacím stroji.

Čárových kódů ale existuje několik a pro výběr toho nejvhodnějšího je nutné znát výhody a nevýhody jednotlivých kódů. V další části jsou tedy jednotlivé typy čárových kódů stručně popsány aby pak mohl být vybrán ten nevhodnější.

#### *3.1.5.1 Teoretický úvod do problematiky*

Čárové kódy spadají do oblasti tzv. "automatické identifikace" neboli jinak řečeno do oblasti "registrace dat bez použití kláves". Do stejné oblasti patří rovněž magnetické kódy používané např. na kreditních kartách nebo strojově čitelné písmo OCR. Čárové kódy jsou z různých důvodů nejrozšířenější. Podrobněji výhody ale i nevýhody čárových kódů budou nastíněny dále.

3.1.5.1.1 Princip čárových kódů (jejich čtení)

Čárový kód se skládá z určitého počtu tmavých čar a světlých mezer, které vyjadřují nějaká čísla popřípadě nějaký text. Kód (čáry a mezery) se čte pomocí snímačů vyzařujících většinou červené světlo. Toto světlo je pohlcováno tmavými čárami a odráženo světlými mezerami. Snímač zjišťuje rozdíly v reflexi a ty přeměňuje v elektrické signály odpovídající šířce čar a mezer. Tyto signály jsou převedeny v číslice, popř. písmena, jaká obsahuje příslušný čárový kód. To tedy znamená, že každá číslice či písmeno je zaznamenáno v čárovém kódu pomocí předem přesně definovaných šířek čar a mezer. Data obsažená v čárovém kódu mohou zahrnovat takřka cokoliv: číslo výrobce, číslo výrobku, místo uložení ve skladu, číslo série nebo dokonce jméno určité osoby, které je např. povolen vstup do jinak uzavřeného prostoru.

3.1.5.1.2 Přednosti čárových kódů

Z mnoha výhod tohoto způsobu uchování informace lze vyzvednout několik následujících:

- Relativně *jednoduchý postup při načítání* čárového kódu (jak z hlediska hardwaru a praktické realizovatelnosti, tak i složitosti softwarového programu). Je totiž výrazně jed-

nodušší vyhodnotit vstup, který se skládá z různě širokých tmavých a světlých míst, než vyhodnotit obraz písmen.

- *Možnost tisku* na papír (či jiné vhodné médium) a tedy i velká variabilita obsahu kódované informace. Není tedy problém přiřadit dvěma po sobě následujícím produktům různé kódy.
- *Přesnost* - užití čárových kódů je jedna z nejpřesnějších a nejrychlejších metod k registraci většího množství dat. Při ručním zadávání dat dochází k chybě průměrně při každém třístém zadání, při použití čárových kódů se počet chyb snižuje až na jednu miliontinu, přičemž většina z těchto chyb může být eliminována, je-li do kódu zavedena kontrolní číslice, která ověřuje správnost čtení všech ostatních číslic.
- *Rychlost* - načtení dat z čárového kódu je nejméně třikrát rychlejší než zadání z klávesnice, nehledě na to, že čtení čárového kódu lze u některých aplikací výhodně s výhodou *automatizovat*.
- *Flexibilita* - technologie čárových kódů je mnohoúčelová, spolehlivá a má snadné užití. Čárové kódy se mohou užívat v nejrůznějších a extrémních prostředích a terénech. Je možné je tisknout na materiály odolné vysokým teplotám nebo naopak extrémním mrazům, na materiály odolné kyselinám, obroušení, nadměrné vlhkosti. Jejich rozměry mohou být dokonce přizpůsobeny tak, aby mohly být užity i na miniaturní elektronické součástky.

V porovnání s ostatními způsoby uchovaní informace je cena paměťového média (papír) a zapisovacího zařízení rovněž velmi výhodná.

#### 3.1.5.1.3 Nevýhody čárových kódů

Na druhou stranu je zřejmé, že zdaleka *ne ve všech případech je možno tisknout* čárový kód. Další nevýhodou je skutečnost, že u tradičních jednodimenzionálních čárových kódů slouží *zakódovaná informace pouze jako klíč* k vyhledání informace v nějaké databázi externího systému. Jednodimenzionální kódy mají tedy nízkou informační kapacitu.

Tuto nevýhodu poněkud odstraňuje nová generace dvojdimenzionálních čárových kódů. Tyto kódy mají řádově vyšší informační kapacitu. Kromě toho si zachovávají schopnost detekce a opravy chyb (při porušení kódu). Podstatnou výhodou je nezávislost na vnějším systému, protože do těchto kódů je možno zakódovat celé údaje jako běžný text, grafiku a i speciální programovací instrukce. Například velikost datového souboru u kódu PDF 417 je až 1,1 kB. Nevýhodou je ale nutnost specielního tisku a především speciálních snímačů. V současné době existuje mnoho standardů jednotlivých firem a dvojdimenzionální snímače umí přečíst pouze jeden nebo dva typy těchto kódů (na rozdíl od jednodimenzionálních snímačů, které umějí číst většinou běžně okolo 20 typů kódů).

Pro identifikaci produktu při jednotlivých operacích na navrhované lince je ale jednodimenzionální kód zcela vyhovující.

#### 3.1.5.1.4 Podmínky pro spolehlivé načtení čárového kódu

Aby byl čárový kód dostatečně čitelný, musí splňovat několik podmínek:

- vhodný poměr odrazivosti světlých a tmavých ploch
- dostatečně široká náběhová oblast
- přítomnost startovacího a koncového znaku
- definovaný kontrolní znak (u některých kódů nepovinně)

*Odrazivost světlých a tmavých ploch* souvisí s vlastnostmi barev, kterými byl čárový kód vytištěn. Pro úspěšné přečtení kódu je důležitá odlišitelnost světlé plochy od tmavé a to především s ohledem na fyzikální vlastnosti snímače, světelné podmínky při snímání a podobně. Pro nej-

lepší čtení je samozřejmě nejlepší bílý podklad a na něm vytištěné černé čárky. Avšak ani tato kombinace není zárukou úspěchu. Důležitá je již zmíněná odrazivost (pokud je například černá barva příliš lesklá, odráží světelné paprsky skoro stejně jako bílý podklad).

Před samotným čárovým kódem musí být podle normy takzvaná "*náběhová oblast*" světlé plochy (takové barvy, jako je oblast mezi čárkami). Pokud není tato plocha dostatečně široká, nemůže snímač určit, odkud má začít dekódovat. Stejné podmínky platí i pro oblast konce kódu, protože snímače mohou číst kód z obou stran (nezáleží tedy na tom, jestli je kód snímán normálně či "vzhůru nohama").

Pro rozlišení typu kódu slouží takzvané *startovací a koncové znaky*. U některých kódů je to sekvence pevně definovaných čárek a mezer, u některých je to konkrétní znak (kód 39 začíná a končí například hvězdičkou). U některých kódů může zase začínat výběrem jedním z několika stanovených znaků. Národní kódy pro zboží mají ještě kontrolní sekvenci uprostřed kódu.

Pro kontrolu správného přečtení kódu je využíván *kontrolní znak*. U některých kódů je tento znak povinný, u některých se může nebo nemusí použít.

#### *3.1.5.2 Přehled vybraných typů čárových kódů*

Podobně jako i v jiných oblastech lidské činnosti, s postupem doby byly vyvinuty mnohé druhy čárových kódů. Každý druh byl původně určen pro jinou oblast. Některé například vznikly pro zdravotnictví, jiné pro aplikace v průmyslu, pro pošty. Z tohoto důvodu má každý kód svůj charakter (například kolik spotřebuje místa na vyjádření jednoho znaku) a také určitou množinu znaků, které je schopen vyjádřit. Ve své podstatě se využívá v praxi jen několik málo druhů.

##### 3.1.5.2.1 EAN, UPC a JAN

Tyto tři kódy vznikly v prvopočátku z národních specifikací a jsou především používány jako kódy pro zboží. Každý z těchto kódů je používán v jiné oblasti světa.

V Evropě je používán EAN 8 a EAN 13, v Americe UPC-A a UPC-E, v Japonsku pak JAN. *Všechny tyto kódy umí vyjádřit jen číslice* a to jen patřičný počet znaků, který je určen konkrétní normou. Ve své podstatě se jedná o podobný princip u každé z těchto norem. Jedná se o to, že v rámci tohoto kódu znamená každá skupina číslic něco jiného.

UPC

Historicky jako první vznikl *Universal Product Code* - univerzální kód výrobků - (U.P.C.) byl úspěšně zaváděn v supermarketech od roku 1973. Je navržen z hlediska jednoznačné identifikace výrobku a jeho výrobce. Jeho symbolika je pevné délky, numerická, souvislá. Každý znak má 4 prvky. UPC verze A se používá k zakódování 12-místného čísla. První číslice je znak systému číslování, dalších 5 je identifikační číslo výrobce, dalších 5 je číslo výrobku a poslední číslice je kontrolní znak. Další, od tohoto kódu odvozené, jsou kódy U.P.C. E0 a U.P.C. E1.

EAN

European Article Numbering (EAN) je nadstavbou U.P.C. Jedná se o nejznámější kód užívaný pro zboží prodávané v obchodní síti. Tento kód může užívat každý stát zapojený do mezinárodního sdružení I.A.N.A. EAN (International Article Numbering Association EAN). Čárový kód EAN dokáže kódovat číslice 0 až 9, přičemž každá číslice je kódována dvěma čárami a dvěma mezerami. Může obsahovat buďto 8 číslic (EAN-8) nebo třináct číslic (EAN-13). První dvě nebo tři číslice vždy určují stát původu (např. ČR má číslo 859), dalších několik číslic (většinou čtyři až šest) určují výrobce a zbývající číslice kromě poslední určují konkrétní zboží. Poslední číslice je kontrolní; ta ověřuje správnost dekódování.

Na obr. 3.1-1 je uveden příklad kódu EAN-13 (s hodnotou "8591111000017") a na obr. 3.1-2 pak příklad EAN-8 (s hodnotou "8591832"). V obou případech předčíslí 859 určuje stát původu Českou Republiku.


*Obr. 3.1-1 Příklad EAN-13*


*Obr. 3.1-2 Příklad EAN-8*

Jak již bylo dříve v části 3.1.5.1 zmíněno, každý kód má definované startovací a koncové znaky. Pro přehlednost jsou tyto pro oba kódy znázorněny na obr. 3.1-3 (EAN-13) a na obr. 3.1-4 (EAN-8). Startovací a koncová sekvence je stejná a navíc je ještě kód touto sekvencí rozdělen v polovině.

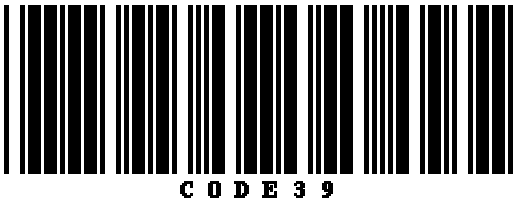

*Obr. 3.1-3 Startovací a koncový znak kódu EAN-13*


*Obr. 3.1-4 Startovací a koncový znak kódu EAN-8*

3.1.5.2.2 Code 39

Tento kód je asi nejvyužívanějším typem čárového kódu. V základní verzi se jedná o kód, který umí kódovat velká písmena, číslice a některé ostatní znaky (-.*$/+%). V rozšířené verzi (Code 39 full ASCII) umí kódovat celou spodní polovinu ASCII tabulky, což jsou malá a velká písmena, číslice a ostatní speciální znaky. Kód 39 full ASCII se kóduje stejně jako normální 39 a pro kódování ostatních znaků je použit normální znak společně s některým speciálním znakem ($ / + %). Na obr. 3.1-5 je opět znázorněna startovací a koncová sekvence, na obr. 3.1-6 pak konkrétní příklad tohoto kódu (s hodnotou "CODE39").


*Obr. 3.1-5 Startovací a koncový znak Code 39*

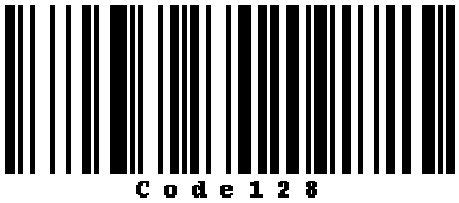



*Obr. 3.1-6 Příklad Code 39*

3.1.5.2.3 Code 128

Tento kód je rozdělen na tři části. Někdy se těmto částem říká Code128A, Code 128B a Code128C. Lze zakódovat celou spodní polovinu tabulky ASCII v základní verzi. To znamená, že je možno vyjádřit každý ze 128 znaků. Každá část však tím samým znakem vyjadřuje něco jiného. TypA obsahuje numerické znaky, znaky velké abecedy, řídicí a speciální znaky, typ B vyjadřuje znaky z ASCII tabulky bez řídicích znaků a speciální znaky. Aby to nebylo tak jednoduché, tak si chytré hlavy usmyslely, že by bylo vhodné ještě vyjádřit číslice nějak hustěji. Tím vznikl typ C, který vyjadřuje jedním znakem vždy dvě číslice (například 00,01,02...). Typ A zase vznikl, aby bylo možno vyjádřit některé speciální netisknutelné znaky.


*Obr. 3.1-7 Startovací a koncové znaky kódů 128 A,B,C*




*Obr. 3.1-8 Příklad kódu 128B*

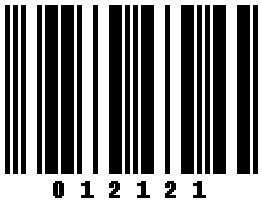
3.1.5.2.4 Kód 2 z 5 interleaved

Tento kód umí vytisknout jen číslice. To proto, že kombinací 5 prvků, z nichž jsou vždy 2 černé není mnoho. Tento kód se hojně využívá tam, kde je nutno zaplnit co nejméně místa vytištěním čárového kódu. Samozřejmě, že v textu mohou být jen číslice.

Specialitou tohoto kódu je, že se vždy kódují dvě číslice do jedné pozice (kód prokládaný-interleaved). Proto má tu výhodu, že vytištěný kód je nejužší ze všech typů. Má to zároveň nevýhodu a to tu, že kód musí obsahovat sudý počet číslic. Proto například některé programy pro tisk čárových kódů nebo tiskárny pro čárový kód doplňují na začátek nulu, pokud je lichý počet číslic.

Na obr. 3.1-9 je znázorněna startovací a koncová sekvence kódu 2/5 interleaved a na obr. 3.1-10 pak příklad tohoto kódu (s hodnotou "012121").


*Obr. 3.1-9 Startovací a koncové znaky kódu 2z5 interleaved*



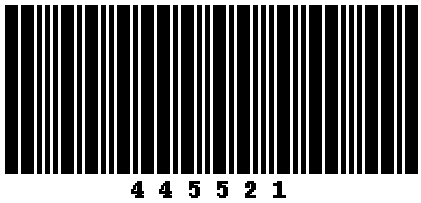

*Obr. 3.1-10 Příklad kódu 2/5 interleaved*

3.1.5.2.5 Code 2/5 industrial

Tento kód je standardem v kódech 2 z 5. Používá se v průmyslových aplikacích, kde se využívají kódy, obsahující jen číslice.

Na obr. 3.1-11 je znázorněna startovací a koncová sekvence kódu 2/5 interleaved a na obr. 3.1-12 pak příklad tohoto kódu (s hodnotou "445521"). Je zajímavé porovnat šířku tohoto kódu s šířkou kódu 2/5 interleaved. Je zřejmé, že přestože vyjadřují stejný počet číslic, kód 2/5 industrial je výrazně širší.


*Obr. 3.1-11 Startovací a koncové znaky kódu 2/5 industrial*



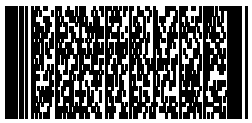

*Obr. 3.1-12 Příklad kódu 2/5 industrial*

3.1.5.2.6 CodaBar

Kořeny tohoto kódu sahají do oblasti zdravotnictví. Původně vznikl pro označování krevních konzerv. Jedná se o kód, který umí vytisknout číslice, písmena A,B,C a D a některé speciální znaky (- $ : / . +). Jeho struktura musí být taková, že začíná a končí vždy písmenem a uprostřed jsou číslice popřípadě vyjmenované znaky. Na začátku nemusí být stejné písmeno jako na konci.

3.1.5.2.7 Ostatní kódy

Existuje ještě celá řada dalších (speciálních) kódů, ty však nemá smysl uvádět v tomto přehledu. Pro zajímavost je na obr. 3.1-13 ještě uveden příklad dvojdimenzionálního kódu (již zmíněný v části 3.1.5.1.3) a to konkrétně kód typu PDF 417.



*Obr. 3.1-13 PDF 417*

### *3.1.5.3 Výběr konkrétního typu kódu*

Pro daný systém je nejvhodnější použít kód 2/5 interleaved a to především díky jeho dobré čitelnosti, dostupnosti čtecích zařízení a vyhovující schopnosti zakódovat dané množství informací.

## *3.2 Návrh realizace z hardwarového hlediska*

Výhody orientace na jednoho výrobce byly již částečně nastíněné v kapitole 2.6.1.3

Firma SRE používá jako řídicí systémy převážně produkty B&R a z tohoto důvodu je zřejmé, že velká část použitých hardwarových komponent - především programovatelné automaty ale i systémy pro řízení servomotorů a podobně - pro navrhovaný systém bude z rodiny produktů firmy Bernecker&Rainer. Je důležité porozumět funkci a možnosti používaných produktů rovněž jako znát širší souvislosti ve vztahu k filosofii firmy a proto je v části 3.2.2 uveden přehled produktů této firmy. Vybrané produkty pro hardwarovou realizaci jsou popsány v části 3.2.3.

Ještě předtím však není od věci se v této souvislosti zmínit v následující části o poněkud odlišném přístupu k programovatelným automatům u firmy B&R obecně.

### 3.2.1 PLC vs. PCC

Standardně bývají programovatelné automaty v angličtině označovány PLC (Programmable Logic Controller - programovatelný logický automat) popřípadě v němčině SPS (Speicher Programmierbare Steuerung - paměťově programovatelné řízení) což zvýrazňuje přístup k programování z hlediska logických funkcí. Proto se u programovatelných automatů různých výrobců většinou setkáváme s možností programovat v následujících jazycích:

- jazyk kontaktových schémat
- jazyk symbolických instrukcí
- jazyk funkčních bloků

Dle velikosti, složitosti (a ceny) PA pak ještě dále mohou být nabízeny další programovací jazyky jako jazyk sekvenčních funkčních diagramů (SFC - Sequential Function Diagram), strukturovaný text, jazyk C a popřípadě další.

Zároveň i struktura a funkce operačních systémů jednotlivých programovatelných automatů různých výrobců a s tím související filosofie programování z hlediska struktury celého programu a jeho vykonávání programovatelným automatem je značně odlišná. Je zřejmé, že pochopit jakým způsobem je celý program svázán dohromady a v jaké návaznosti jsou jednotlivé části programu vykonávány, je základním předpokladem pro správné řešení dané automatizační úlohy.

Zmiňovaná odlišnost přístupu k programovatelným automatům spočívá právě ve struktuře operačního systému a s tím související hardwarové struktury. B&R nazývá své automaty PCC (Programmable Computer Controller) respektive RPS (Rechner Programmierbare Steuerung - počítačově programovatelné řízení). Rozdíl oproti klasickému PLC spočívá v možnosti použití vyšších programovacích jazyků, časově paralelnímu provádění více programů na jedné CPU (procesoru) a paměťové kapacitě až v řádech megabytů. Podrobněji je struktura operačního systému automatů B&R popsána v odstavci 3.2.2.4.

### 3.2.2 Přehled produktů firmy Bernecker&Rainer (B&R)

#### *3.2.2.1 Dělení produktů B&R do skupin*

Tak jako ostatní firmy nabízející komplexní řešení pro automatizaci, i B&R dělí své produkty do jednotlivých skupin.

Jsou to:

- Automation Software
- Automation Net
- Automation Runtime
- Automation Targets

Do skupiny *Automation Software* patří vývojové prostředí Automation Studio pro programování různých cílových systémů.

*Automation Net* umožňuje komunikaci mezi jednotlivými navzájem propojenými účastníky, ať již se jedná o B&R komponenty propojených sběrnicemi nebo o jiné připojené produkty.

*Automation Runtime* je operační systém a představuje mezičlánek mezi skupinami Automation Software a Automation Targets. Ke spojení je používána právě Automation Net.

Pod název *Automation Target* je zahrnut hardware, na kterém běží operační systém Automation Runtime. Tuto skupinu je možno ještě dále rozdělit na následující podskupiny:

- Control Systems - řídicí systémy od nejmenších až po největší
- Panel Systems - od kompaktních operátorských panelů až po displeje s modulárními IPC
- Motion Systems - pohony a motory pro dynamické a precizní polohování

#### *3.2.2.2 Automation Studio*

Toto prostředí má podobné vlastnosti jako ostatní vývojová prostředí pro programovatelné automaty jiných firem.

Z hardwarového hlediska je Automation Studio nainstalované na PC schopno po připojení přes rozhraní RS 232 popřípadě Ethernet detekovat konfiguraci připojeného programovatelného automatu. Dále je zde možno výběrem z dialogových oken vybrat jednotlivé rozšiřující komponenty. Pro snazší orientaci je Automation Studio vybaveno i interaktivním popisem jednotlivých komponent.

Co se týče Softwarové stránky, je v Automation Studio možno psát a ladit programy pro všechny automaty firmy B&R. Jádrem celého systému je samozřejmě možnost tvorby programů pro řízení procesu stroje, pro který je automat určen.

Programovat je možné ve většině standardních jazycích:

- Automation Basic
- ANSI-C
- jazyky podle IEC 61131-3
  - kontaktní schémata (LAD)
  - jazyk mnemokódů (IL)
  - strukturovaný text (ST)
  - sekvenční vývojové diagramy (SFC)

Dále je k dispozici editor datových modulů a editor datových typů

Zde je patrně zajímavé podotknout, že tento výběr programovacích jazyků platí pro všechny cílové platformy B&R.

Kromě toho je v Automation Studiu integrováno ještě prostředí pro *vizualizaci* řízení a stavu procesu (Visual Components). Podstatnou výhodou je možnost použití stejných proměnných používaných v programu a tím tedy značné zjednodušení především s ohledem na fázi vývoje programu a vizualizace. Nedostatkem je poněkud horší uživatelský komfort (chybí například knihovna vizualizačních symbolů).

Editor procesních obrazovek s plnou podporou grafiky se automaticky adaptuje na používaný hardware. Specifikace možností Visual Components:

- 256 barev (internetový standard)
- rastrové obrázky, text, čáry, obdélníky, elipsy, kruhy a vyplněné oblasti s dynamickými změnami barev
- vstupní a výstupní pole (číselná, alfanumerická, sloupcové grafy, rastrové obrázky)
- tlačítka, aktivní oblasti, seznamy, skupiny textů
- dynamické operace (zpracování hodnot, barev a atributů)
- změny jazyka online (text, formát atd.)

- předdefinovaná písma ve třech velikostech nebo písma TrueType
- 4 významové úrovně klávesnice
- číselná klávesnice nebo klávesnice na dotykové obrazovce
- režim tipování
- programování nestandardních grafických výstupů funkčními bloky VISAPI

#### *3.2.2.3 Automation Targets*

Jak již bylo uvedeno, pod tímto názvem se rozumí hardwarová platforma, na které běží operační systém Automation Runtime.

Existují tři hlavní skupiny lišící se vlastnostmi a použitím: Control Systems, Panel Systems a Motion Systems.

3.2.2.3.1 Control Systems (Řídicí systémy)

V této podskupině jsou zahrnuty jak *programovatelné automaty* tak i *průmyslové počítače*.

*3.2.2.3.1.1 B&R 2000*

Programovatelné automaty jsou představovány typovou řadou B&R 2000. Tato řada je ještě dále členěna dle výkonnosti (velikosti):

B&R 2003
B&R 2005
B&R 2010

Z hardwarového hlediska je možno B&R 2000 stručně charakterizovat následovně:

- modulární struktura
- možnost připojení na sběrnici (CAN-Bus, Powerlink, X2XLink, Ethernet)
- komunikační rozhraní pro připojení HMI
- bezpečný I/O Bus Protocol
- vzdálené I/O body (RIO)
- zpracování bitů nebo slov v rámci jednoho cyklu

Softwarová charakteristika pak vypadá následovně:

- kvazi-multitasking
- možnost použití vyšších programovacích jazyků
- přesná kontrola doby cyklu

*3.2.2.3.1.2 IPC*

Pro použití v oblasti průmyslových počítačů (IPC) jsou k dispozici dvě odlišné typové řady - SlotPLC (Logic Scanner) a SoftPLC (Provit).

Řada **Logic Scanner** je vyráběna jako PCI-Bus karta, kterou je možno zasunout do jakéhokoli počítače (a tedy i průmyslového) vybaveného PCI slotem (zdířkou). Tato karta v podstatě představuje programovatelný automat. Připojení vstupů a výstupů může být realizováno pomocí sběrnic CAN nebo Remote Input/Output (RIO) (Popřípadě X67). Karta má svůj vlastní operační systém patřičné verze Automation Runtime a je tak zcela nezávislá na stavu operačního systému hostitelského počítače. Díky tomu je možno docílit stabilního, real-time řízení cílového procesu s kvazi-multitaskingem. Na PC pak může pod vhodným operačním systémem probíhat vizualizace daného procesu.

V řadě **Provit** jsou zastoupeny dva typy průmyslových počítačů - IPC 2xxx a IPC 5xxx. Typy IPC 2xxx v podstatě umožňují realizovat kombinací ovládacího panelu a programovatelného automatu, ale navíc nabízejí možnosti (výkon a otevřenost) průmyslového počítače.

Typy IPC 5xxx se vyznačují větší modularitou, vyšším výkonem a na rozdíl od IPC 2xxx displeje nejsou přimontovány přímo na skříni IPC.

Kromě kombinace běžně používaných operačních systémů a karty Logic Scanner je zajímavá možnost emulace programovatelného automatu přímo na IPC (také nazýváno SoftPLC). Děje se tak s pomocí příslušné verze operačního systému Automation Runtime. Tímto způsobem je zaručena práce v reálném čase spolu. IPC s tímto operačním systémem je pak možno standardně programovat pomocí Automation Studia.

3.2.2.3.2 Panel Systems

Prostředky HMI pro ovládání člověkem a vizualizaci firmy B&R jsou zahrnuty v typových řadách PANELWARE a Provit. Řada PANELWARE je určena pro spíše menší vizualizační úlohy, displeje z řady Provit je možno použít i na náročné vizualizace.

*3.2.2.3.2.1 PANELWARE*

Jedná se o modulární koncepci ovládacích panelů s klávesovými bloky, displejů a programovatelných automatů.

*3.2.2.3.2.2 Provit*

Řada IPC 2xxx je určena spíše pro vizualizace střední náročnosti, IPC 5xxx pak umožňuje realizovat i složité vizualizace. Kromě dotykových displejů či displejů s ovládacími panely je možno připojovat přes sběrnice i komponenty z řady Panelware. (Například hlavní vizualizace a ovládání procesu probíhá na dotykovém displeji, jemné nastavení polohovací osy je možno provést pomocí ovládacího panelu s alfanumerickým displejem přímo v blízkosti osy.)

3.2.2.3.3 Motion Systems (Motory a pohony)

Nezanedbatelnou výhodou polohovacích prostředků B&R je plná integrace do celého systému a tedy i do vývojového prostředí Automation Studio.

Motion Components je možno použít pro aplikace jako polohování, převodovky a elektronické vačky. V Automation Studiu lze v jednom projektu formou objektů a parametrů spravovat až 255 os. Rozsáhlá databáze motorů usnadňuje výběr požadovaného pohonu.

V rámci strojů linky je zajímavý především systém Acopos pro řízení servomotorů, umožňující realizovat složité funkce jako elektronické vačky či synchronizaci více os pomocí virtuální referenční osy a podobně. Detailnější popis tohoto produktu ale není možný z důvodu omezeného rozsahu této práce.

### *3.2.2.4 Automation Runtime*

Operační systémy používané pro jednotlivé programovatelné automaty B&R se patrně liší ve vnitřní struktuře v závislosti na konkrétním hardwaru programovatelného automatu, navenek však mají stejné charakteristické vlastnosti. Jedná se o deterministický (kvazi-) multitaskingový systém, upravený specielně pro potřeby řídicí techniky.

U standardních multitaskingových systémů, které nejsou určeny pro řízení v reálném čase, je otázka doby cyklu jednotlivých úloh přinejmenším problematická a časový průběh jednotlivých úloh může být většinou ovlivněn pouze stanovením priorit jednotlivých úloh. Proto pro řízení v reálném čase je nutno použít odlišný přístup.

Protože změny ve verzi operačního systému a hlavně parametrizaci je možno provádět ve vývojovém prostředí Automation Studio, jsou dále vlastnosti operačního systému popsány v úzké souvislosti s tímto vývojovým prostředím (a tedy vlastně v souvislosti s softwarovou konfigurací).

V editoru projektu Automation Studia je uvedena stromová struktura hardwarové konfigurace aktuálního projektu. Důležitým (a vždy přítomným) prvkem stromu je příslušná CPU. Kromě jiných vlastností je zde možno na záložce Software vytvořit objekty následujících typů:

- cyklické úlohy
- necyklické úlohy
- datový modul
- systémový modul
- rozšířený objekt

Z hlediska operačního systému jsou důležité objekty právě prvního typu - cyklické úlohy ("Cyclic tasks"). Sem jsou zařazeny programy, které má operační systém (programovatelný automat) provádět. Uživatel si může nastavit dobu cyklu, v jaké se má příslušný program cyklicky vykonávat. Počet různých dob cyklu je ale omezen. K dispozici je pět respektive šest "cyklických tříd" (v závislosti na cílovém hardwaru). Úkolem operačního systému je pak zajistit zpracování všech úloh v příslušném časovém limitu.

Kromě doby cyklu je možno u jednotlivých cyklických tříd mimo jiné ještě nastavit časovou toleranci a tím je umožněna kontrola nad skutečným časovým průběhem provádění všech úloh.

Doba cyklu je totiž hodnota nastavená uživatelem pro jednotlivé cyklické třídy. Doba cyklu ale nepodává informaci o skutečné době průběhu, která je potřeba pro vykonání všech úloh obsažených v patřičné cyklické třídě. Definujeme-li skutečnou dobu průběhu úlohy *i* obsažené v cyklické třídě *x* jako $Tx_i$, nastavenou dobu cyklu třídy *x* jako $T_{cyklu}x$ a její toleranci $\tau$, musí pak pro všechny cyklické třídy *x* platit následující vztah:

$$\sum Tx_i \leq T_{cyklu}x + \tau \qquad (3.2\text{-}1)$$

Operační systém Automation Runtime je možno nastavit tak, aby v případě, že zjistí, že podmínka (3.2-1) nebyla splněna při vykonávání programu, nahlásil chybu.

Hlavní vlastnosti, které splňuje popsaný (kvazi-)multitaskingový operační systém lze tedy shrnout do následujících bodů:

- paralelní zpracování více úloh
- deterministický multitasking
- rozdílné a nastavitelné doby cyklu s možností kontroly
- časově konzistentní čtení/zápis jednotlivých vstupů/výstupů pro každou třídu dobu cyklu.

#### *3.2.2.5 Automation Net*

Komunikační platformu Automation Net používají všechny součásti systému B&R Automation Software. Všechny stanice komunikující v síti si mohou vyměňovat a zpracovávat programové objekty a procesní data nezávisle na cílovém systému, používaném přenosovém médiu a protokolu.

Platforma Automation Net využívá standardní rozhraní Windows a umožňuje systémům B&R propojení s kancelářskými nástroji.

"PVI Controls" je rozhraní ActiveX pro programátory v jazycích Visual Basic a VBA. Rychlý a jednoduchý provoz - proměnné PLC lze bez dalšího programování zobrazovat jejich "propojením" se standardními ovládacími prvky Visual Basic.

Integrovaný webový server sítě Automation Net a operační systém Automation Runtime umožňuje zobrazování procesních proměnných v libovolném webovém prohlížeči.

| | |
|---|---|
| Médium | sériové rozhraní RS232, modem<br>CAN až do rychlosti 500 kBd<br>Ethernet TCP/IP 10/100 Mbps |
| Rozhraní | DLL (PVI, Process Visualization Interface)<br>server OPC, server DDE, webový server |

### 3.2.3 Vybrané produkty pro hardwarovou realizaci

Vzhledem k rozsahu řešení softwarového návrhu a rovněž v důsledku zpoždění konstrukčních prací na jednotlivých strojích není konkrétní návrh hardwarové realizace stěžejním bodem této práce.

Přesto však jsou v následující části uvedeny možné varianty nasazení hardwarových komponent jak pro řídicí systém tak pro jednotlivé programovatelné automaty.

#### *3.2.3.1 Řídicí systém*

Jádrem řídicího systému bude nepochybně průmyslové PC. Z hlediska komunikace pak musí být vybaveno ethernetovým rozhraním. Výrobce může být různý, v případě B&R ale odpadá nutnost kupovat licenci pro B&R PVI OPC Server, neboť ta není nutná v případě hardwaru B&R.

Softwarový modul webové aplikace pak může být spuštěn buď rovněž v rámci tohoto IPC, výhodnější z hlediska výkonu a bezpečnosti (ale také dražší) je pak hardwarové oddělení , tedy provoz webové aplikace na jiném počítači spojeném přes Ethernet s IPC. Pro webový modul pak může být použito „obyčejné“ PC.

#### *3.2.3.2 Programovatelné automaty*

Výběr PA pro stroje je omezený z důvodů orientace firmy SRE na B&R. Dle způsobu, jakým jsou stroje ve firmě konstruovány, přicházejí v zásadě v úvahu dvě varianty konfigurace:

V první variantě automat stroje představuje **IPC** (SoftPLC s Automation Runtime) řady Provit a napojený dotykový panel a dále přes CAN napojené moduly. IPC má standardně mimo jiné Ethernetové rozhraní a je možná komunikace s řídicím systémem (respektive B&R PVI OPC serverem).

V druhé variantě je pro mechanické řízení stroje použita CPU řady 2003 s přídavnými moduly a pro vizualizaci a správu zakázky PowerPanel. Ten je vybaven Ethernetovým rozhraním a může opět komunikovat s OPC serverem.

## *3.3 Návrh realizace ze softwarového hlediska*

V této části je uvedeno řešení nejvyšší vrstvy řízení výrobního systému – webové aplikace pro správu zakázek, které využívá dispečerská úroveň řízení výrobního systému jako zdroj dat pro organizaci a parametry výroby.

Vlastní softwarové řešení dispečerské úrovně řídicího systému je komplexní a provázané s komunikací s podřízenými systémy a je tedy uvedeno ve samostatné kapitole 4.

### 3.3.1 Webová aplikace pro správu zakázek výrobního systému (založená na práci s s JDF soubory)

Jak již bylo zmíněno v části 3.1.1.4, pro realizaci správy zakázek výrobního systému a práce s JDF soubory jsem zvolil řešení založené na severu Apache využívajícím modul PHP pro dynamické generování stránek určených pro prohlížeč a práci se soubory XML rovněž jako pro přístup k databázím MySQL a MS Access.

Návrh aplikace je možno rozdělit na dvě (spolu související) hlavní části:

- návrh obecného schématu editace JDF souborů z uživatelského hlediska
- návrh postupu při vytvoření a editaci JDF souborů z hlediska programové realizace a ohledem na omezení plynoucí z podstaty procesu dokončovacích operací.

Pro řešení první části z uživatelského hlediska se jevilo jako nejvýhodnější navrhnout rozhraní podobné standardním aplikacím pro práci se soubory. Důvody jsou zřejmé, neb v tomto případě může uživatel pracovat s JDF soubory víceméně intuitivně. Soubor vytvořených PHP skriptů a HTML kódu realizující tuto část jsem pojmenoval JDF File Explorer. Stručný popis je uveden v 3.3.1.1.

Podstatou druhé části je především editace JDF souborů přístupných z JDF File Exploreru. V zásadě se jedná o načtení parametrů z JDF souboru (tedy atributů jednotlivých elementů XML souboru), jejich editace uživatelem a opětovné uložení do JDF souboru. Vzhledem k charakteru procesu je nutná kontrola korektnosti parametrů před jejich uložením. Všechny tyto činnosti jsem shrnul pod část aplikace nazvanou JDF File Editor, která je podrobněji popsána v 3.3.1.2.

Zmíněné dvě hlavní části návrhu je nutné ještě doplnit o následující:

- *Podpora více uživatelů v rámci jedné aplikace*. Opět jsem zvolil standardní řešení spočívající v existenci uživatelského jména a hesla, po jejichž zadání v přihlašovacím dialogu je načten příslušný domovský adresář (home) specifický a jedinečný pro každého uživatele.
- *Doplnění zámku* po otevření *JDF souboru pro editaci* a opětovné odemknutí při uložení daného souboru (s časovou kontrolou a opětném odemknutí v případě delší nečinnosti)

Pro realizaci podobných úkolů je vhodné doplnit stávající dvojici Apache a PHP o databázový systém MySQL.

Pro ostrý provoz v rámci internetu a návazností na výrobní systém (odesílání zakázek k výrobě) je pak samozřejmě nutné realizovat přenos ne pomocí standardního protokolu http, ale pomocí zabezpečeného https. Jeho aplikace ale není součástí této diplomové práce.

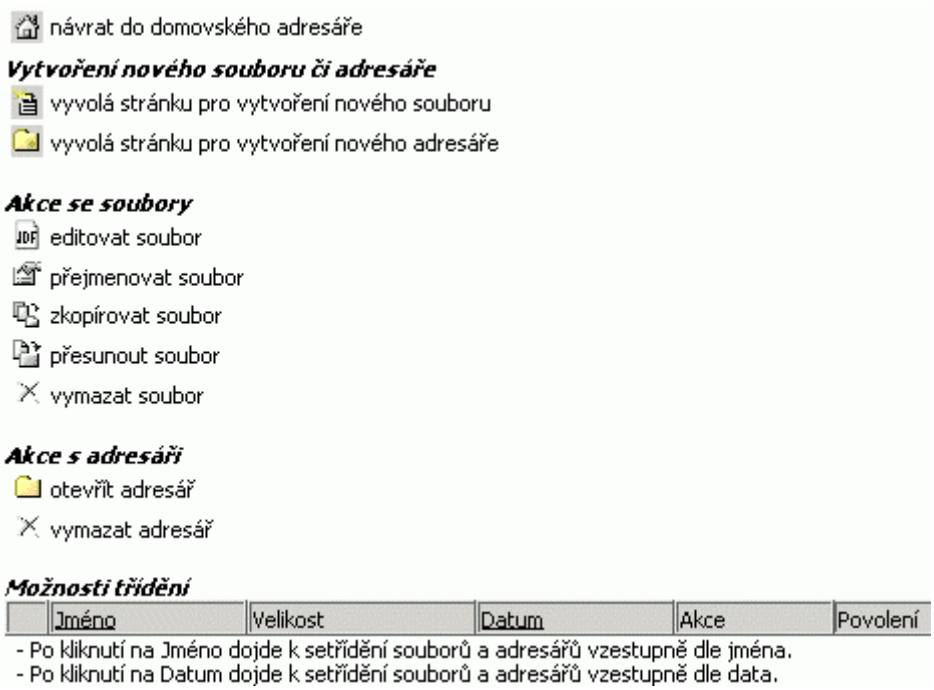
Dále jsou podrobněji popsány vybrané části webové aplikace. Kvůli omezenému rozsahu této práce není bohužel možné popsat všechny funkce a vlastnosti této webové aplikace. Kompletně má zdrojový text této aplikace (HTML + PHP) více než 12 000 řádků.

#### *3.3.1.1 JDF File Explorer*

JDF File Explorer se zobrazí po zadání uživatelského jména a hesla a vstupu do systému. Je realizován PHP skriptem index.php.

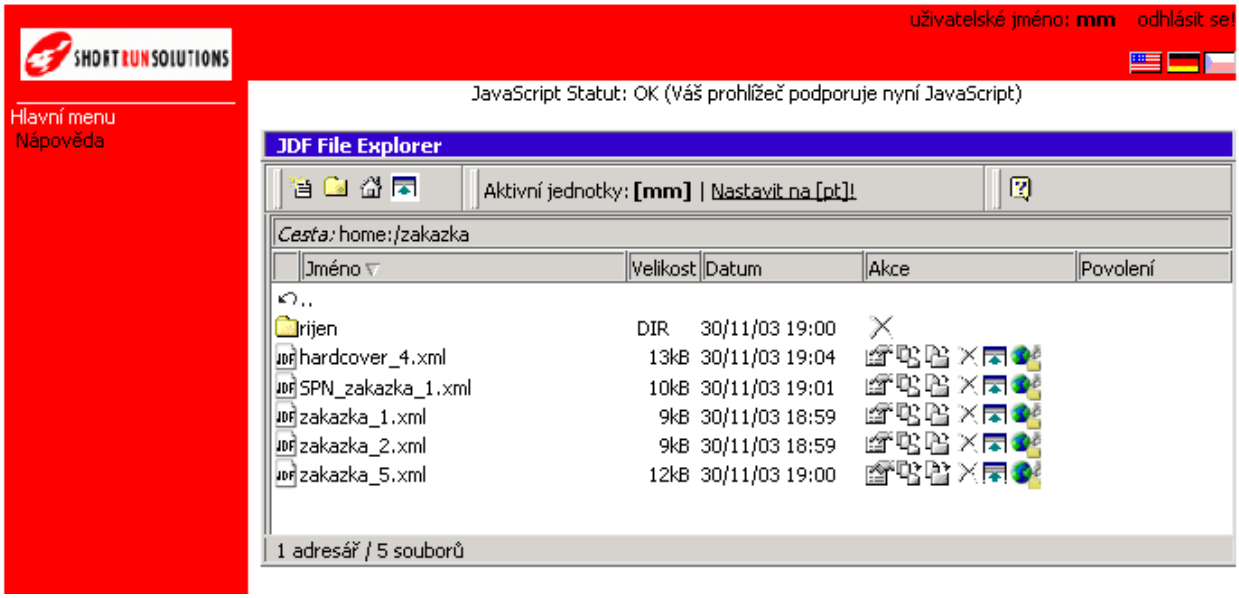
Pro každého uživatele je tedy vytvořen jeho vlastní virtuální domovský adresář (home). Domovské adresáře jsou na uloženy na disku, kde běží webová aplikace. Uživatelé mají práva pro vytváření adresářů a souborů ve svém vlastním domovském adresáři (tedy směrem dolů), ale nemohou přistupovat k nadřazeným adresářům.

Základní funkce z uživatelského hlediska jsou patrné z následujícího obrázku, který byl zkopírován z nápovědy, kterou jsem vytvořil k JDF File Exploreru.:




*Obr. 3.3-1 Výřez z nápovědy k JDF File Exploreru*

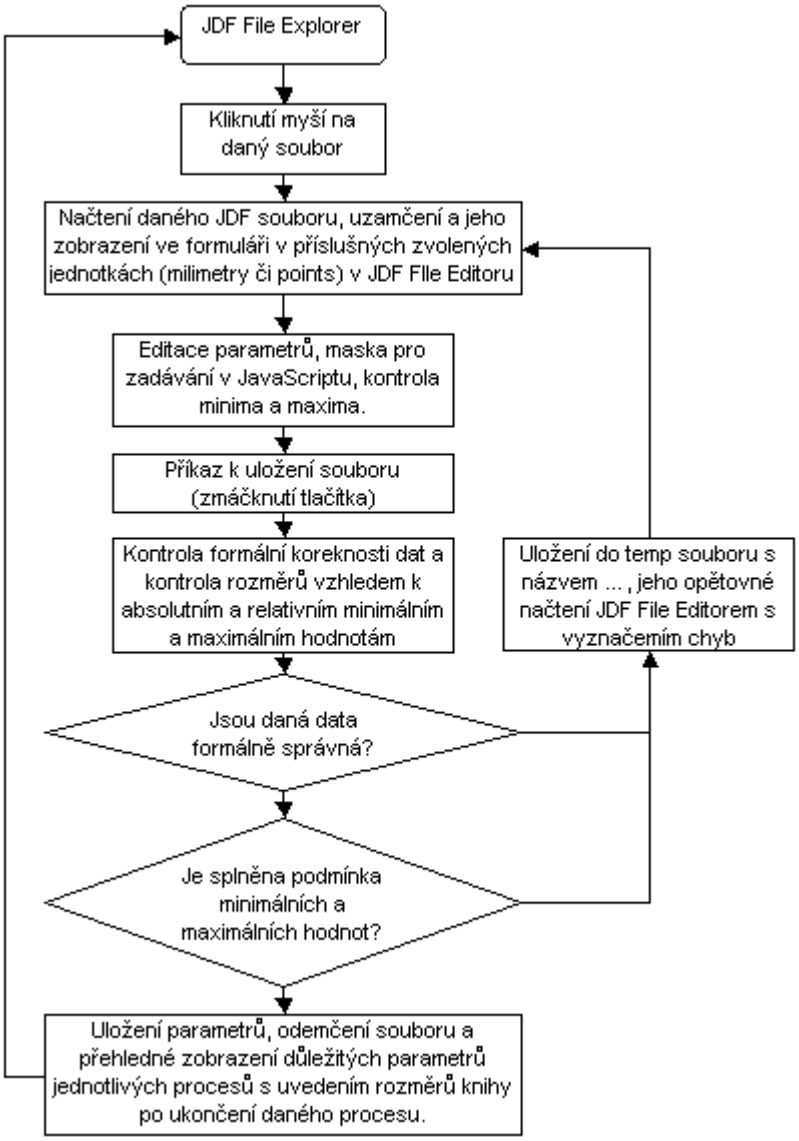
Pro názornost je rovněž téměř pro všechny operace se soubory u jednotlivých ovládacích prvků kontextová nápověda automaticky aktivovaná po najetí myší na daný ovládací prvek. Na obr. 3.3-2 je uvedena ukázka obrazovky JDF File Exploreru.




*Obr. 3.3-2 JDF File Explorer – ukázka obrazovky*

#### *3.3.1.2 JDF File Editor*

Postup při editování JDF souboru JDF File Editorem je zobrazen na obr. 3.3-3.

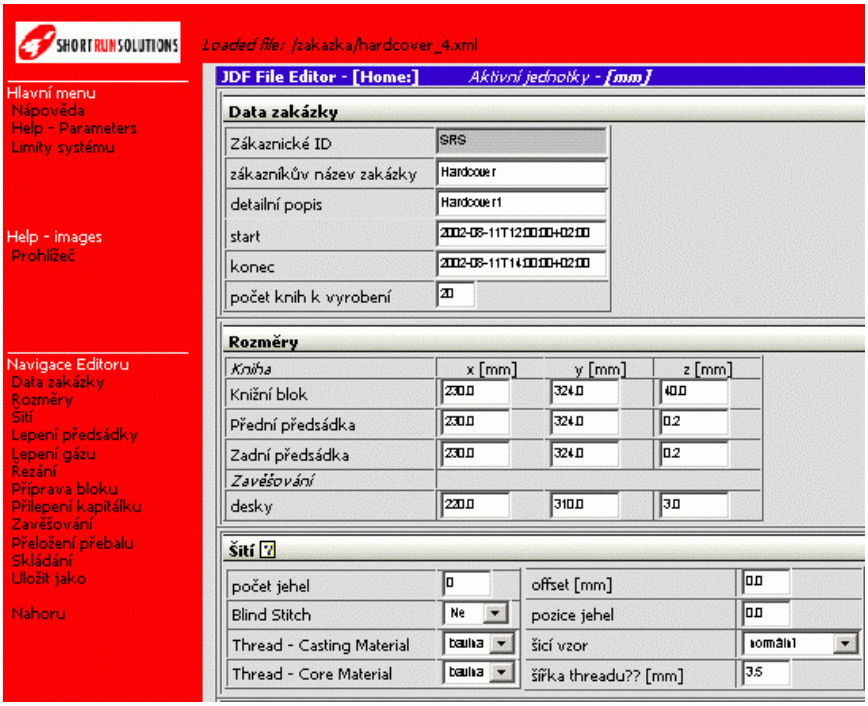



*Obr. 3.3-3 Vývojový diagram – JDF File Editor*

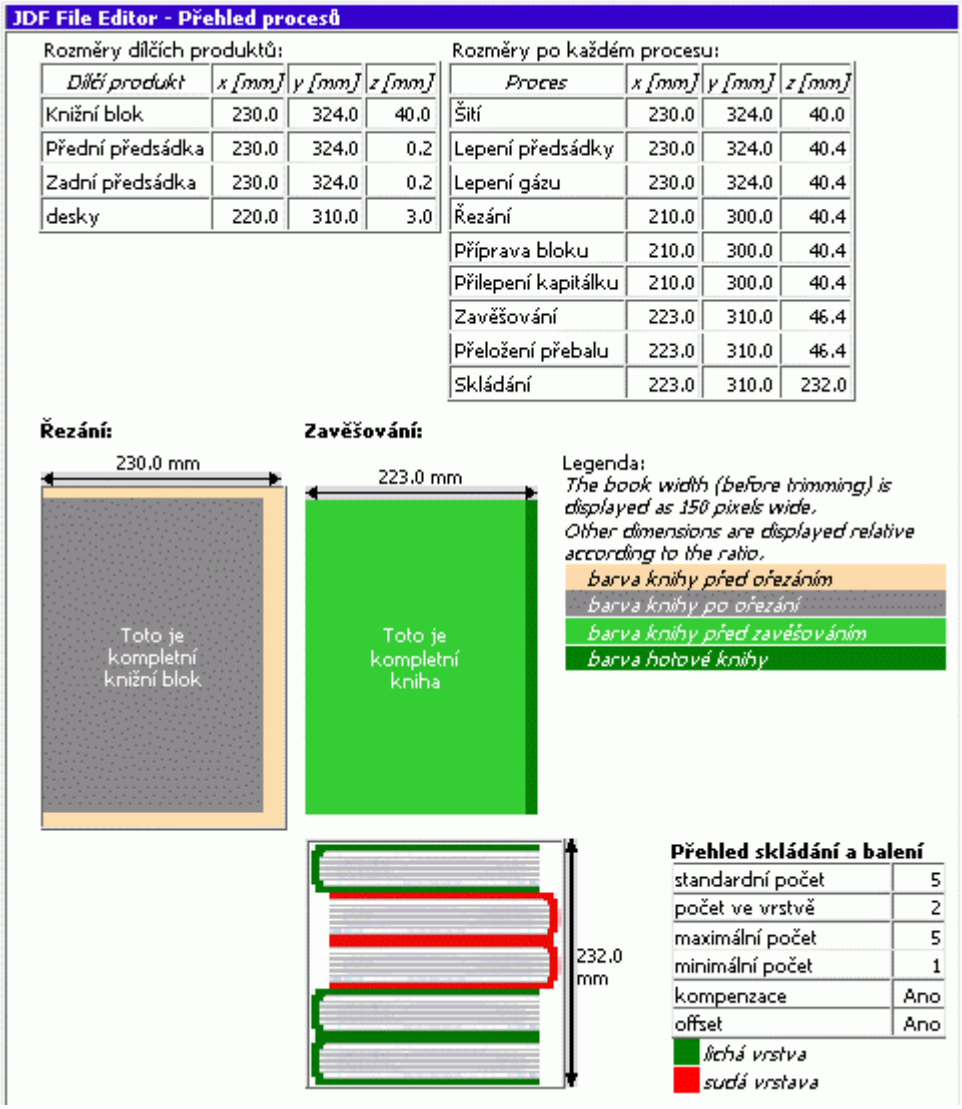
Tedy hlavní body editace JDF souboru lze shrnout:

1) Otevření souboru z JDF FIle Exploreru pro editaci (kliknutí myší)
2) Načtení zvoleného JDF/XML souboru, jeho uzamčení (read-only) a zobrazení ve formuláři
3) Zobrazení dat ve zvolených jednotkách buď milimetry [mm] nebo bodech (points) [pt]
4) Maska pro zadávání dat a kontrola realizovatelnosti dat (dle aktuální konfigurace linky) na straně klienta řešená v jazyce Javascript
5) Kontrola zadaných dat (formální správnosti a realizovatelnosti) na straně serveru pomocí PHP skriptu.
6) V případě správnosti uložení, odemčení a zobrazení parametrů jednotlivých procesů tak jak jdou po sobě
7) Návrat do JDF File Exploreru

Na obr. 3.3-4 je zobrazena část obrazovky při editaci souboru „hardcover_4.xml“ z podadresáře „zakazka“.




*Obr. 3.3-4 JDF File Editor – ukázka části obrazovky*




*Obr. 3.3-5 JDF File Editor- přehled procesů - upravený výřez*

Přehled procesů po jeho úspěšném uložení je zobrazen na obr. 3.3-5. Tento přehled rovněž uvádí grafický přehled rozměrů důležitých dílčích produktů, jejichž proporce odpovídají parametrům zobrazené zakázky a umožňují tak určitou kontrolu pro uživatele, že zadaná data odpovídají jeho představám.

#### *3.3.1.3 Kontrola dat*

Jak již bylo dříve zmíněno, je nutné v rámci JDF File Editoru kontrolovat ukládaná data. A to principielně ze dvou hledisek:

- *formální korektnost dat* (tzn. například že pro údaje typu číslo s možností desetinné čárky bude tento parametr zadaný uživatelem skutečně obsahovat pouze číslice a maximálně jednu desetinnou čárku či například údaj o maximálním počtu knih naskládaných v jednom balíku bude obsahovat pouze číslice)
- *maximální a minimální hodnoty* – neboť pro parametry dokončovacích operací existují minimální a maximální hodnoty specifické pro danou výrobní linku.

Podrobněji je kontrola dat popsána v následující části a to jak kontrola na straně klienta (pomocí JavaScriptu) tak i na straně Serveru (pomocí PHP). Důvodem této duplicity je snaha o maximální podporu uživatele. Kontrola na straně klienta je do jisté míry redundantní, ale jednak urychluje práci uživatele, který nemusí čekat během doby odeslání dat na server, jejich kontrolu a případné odeslání zprávy, že data nejsou správná. Na druhou stranou především při větším počtu současně pracujících uživatelů dochází při absenci kontroly na straně klienta ke zbytečnému zatěžování serveru.

V každém případě není možné kontrolu na straně serveru vynechat, neboť se nelze obecně spoléhat na kvalitu dat zkontrolovaných pouze na straně klienta a odeslaných na server. A to v prvé řadě principielně z bezpečnostních důvodů (například při přijímání dat z formuláře, která budou použita pro práci s databází) nebo i jen z toho důvodu, že uživatel má deaktivovaný JavaScript. (Přestože je tato skutečnost testována).

##### 3.3.1.3.1 Kontrola na straně klienta (Client-Side) pomocí Javascriptu

###### *3.3.1.3.1.1 Maska pro zadávání dat*

V tab. 3.3-1 jsou uvedeny základní typy polí a znaky, ze kterých se mohou sestávat jejich parametry.

| *Typ pole* | *Dovolené znaky* | *Poznámka* |
|---|---|---|
| Rozměry | '0..9' a '.' | Např. Trimmed Width<br>*JavaSript funkce numericAD* |
| Údaje o počtu | '0..9' | Např. Layer Amount<br>*JavaSript funkce numeric* |
| Text | '0..9' a 'A-z' | Např. Glue Brand<br>*JavaSript funkce alfanumeric* |
| Název souboru | '0..9' a 'A-z' a '_' | Znaky které smí obsahovat jméno soubouru<br>*JavaSript funkce alfanumericAC* |

*Tab. 3.3-1 Typy polí a dovolené znaky masky pro zadávání dat*

Maska je realizována v jazyce JavaScript:

Kód JavaScript:

```
function getkey(e)
{
  var code;
  if (!e)
    var e = window.event; //
nastaveni pro IE
  if (e.keyCode)
```

```
function alfanumeric(eX,diacritic)
{
  test=getkey(eX);
  set1=(test>32 && test<48);
  set2=(test>57 && test<65);
  set3=(test>90 && test<97);
  if (diacritic == 1)
```

```
    code = e.keyCode; // ie and
mozilla/gecko
  else
   if (e.which)
     code = e.which; // nn4
  return code;
}
function numeric(eX)
{
  test=getkey(eX);
  if (test<48 || test>57)
    return false;
}
function numericAD(eX)
{
  test=getkey(eX);
  set1=(test<46 && test!=8 &&
test!=9);
  set2=(test>46 && test<48);
      //46='.'
  set3=(test>57);
  if (set1 || set2 || set3)
             return false;
}
```

```
    set4=(test>122 && test<127);
  else
    set4=(test>122)
  if (set1 || set2 || set3 || set4)
    return false;
}
function alfanumericAC(eX)
      //alfanumeric + ' ' + '_'
{
  test=getkey(eX);
  set1=(test>32 && test<45);  //'
'..-
  set6=(test>45 && test<48);  //'-
'..0'
  set2=(test>57 && test<65);  //9..A
  set3=(test>90 && test<95);  //Z.._
  set4=(test>95 && test<97);  //_..a
  set5=(test>122);
      //..z
  if (set1 || set2 || set3 || set4 ||
set5 || set6)
    return false;
}
```

Příslušná funkce je pak spuštěna v události onKeypress, jak je patrno z následujícího úryvku kódu:

```
x:<input name="Comp_Res12_x" value="23.0" size="10" onkeypress="return
numericAD(event);" onblur="check_limits(this,10,28.5)">
```

*3.3.1.3.1.2 Kontrola krajních mezí dat a realizovatelnosti*

Jsou zde navrženy dva typy parametrů:

a) parametry nezávislé na ostatních zadaných údajích
b) parametry závislé na ostatních zadaných datech.

Data typu a) jsou po opuštění editovaného pole testována oproti jejich maximální a minimální přípustné hodnotě. Tyto hodnoty jsou definovány pro konkrétní konfiguraci výrobní linky a jsou nazvány „System Limits“ – „Mezní hodnoty systému“.

Příklad ad a)

PHP script:

```
x:<input name="Comp_Res12_x" value="<?php
echo(conv_dim($Comp_Res12_Array[0],$act_unit))?>" size="10"
onkeypress="return numericAD(event);" onblur="check_limits(this,<?php
echo($lim['BlockWidth_min'].",".$lim['BlockWidth_max']);?>)">
```

HTML Output:

```
x:<input name="Comp_Res12_x" value="23.0" size="10" onkeypress="return
numericAD(event);" onblur="check_limits(this,10,28.5)">
```

Data typu b) jsou po opuštění editovaného pole stejně jako typu a) testována oproti jejich maximu a minimu. Toto minimum a maximum je ale vždy nově vypočteno na základě hodnoty parametru (konkrétním poli), na kterém je hodnota editovaného pole závislá (to určuje relativní minimum a maximum) a hodnotě absolutního minima a maxima pro editované pole definovaných v mezích systému.

Příklad ad b)

Při zadávání šířky knihy po ořezání jsou v „Systém Limits“ například definovány hodnoty minima a maxima odřezávaného kusu na minimum = 1 cm a maximum = 10 cm. Velikost odřezávaného kusu je ale závislá nejen na konečné šířce knihy po ořezání, ale i na šířce knihy před

ořezáním. Tu je tedy nutno vzít při kontrole v úvahu a na základě těchto údajů je pak možno vypočítat relativní minimum a maximum.

PHP Script

```
<input name="TrP_Width" value="<?php echo(conv_dim($TrP_Width,$act_unit))?>"
size="10" onkeypress="return numericAD(event);"
onblur="check_limits4(this,this.form.Comp_Res12_x,<?php
echo($lim['M_TrimmingWidth_min'].",".$lim['M_TrimmingWidth_max'])?>)">
```

HTML Output

```
<input name="TrP_Width" value="21.0" size="10" onkeypress="return
numericAD(event);" onblur="check_limits4(this,this.form.Comp_Res12_x,1,10)">
```

3.3.1.3.2 Server-Side (Kontrola zadaných dat pomocí PHP)

Jak již bylo dříve zmíněno, kontrola dat na straně serveru pomocí PHP je rozdělena do dvou hlavních částí:

- Kontrola formální správnosti dat
- Kontrola mezních hodnot

Kontrola formální správnosti dat je realizována tak, že jednotlivá pole jsou testována, zda předávané parametry jsou typu, který je pro dané pole určen.

Kontrola mezních hodnot je založena na podobném principu jako již bylo vysvětleno v části kontroly dat na straně klienta s tím rozdílem, že je realizována skriptem v PHP.

### 3.3.2 Řídicí systém (nadřazený počítač)

Úlohou řídicího systému je kromě zprostředkování specifikace vstupních dat především na základě těchto dat řídit těmito daty specifikovaný proces dokončovací výroby.

Primární úkol tak spočívá v předání informace jednotlivým strojům o parametrech právě vyráběné knihy a řídit předávání jednotlivých knih mezi stroji.

Pro efektivní provoz takové linky, umožňující vyrábět knihy o malém nákladu, je ale nejdůležitější co možná nejsamostatnější řešení problémů vzniklých při výrobě. Neboť ani sebelepší konstrukce strojů a jakkoli sofistikovaný program nemůže předvídat všechny problémy vzniklé prací s tak proměnlivým a různorodým materiálem (jako je papír) používaným při výrobě knihy. Právě automatická identifikace a řešení problému, která je možná v případě sledování produktu v kritických místech, pak umožňuje kontinuální provoz bez zásahu lidského faktoru.

# 4 Řídicí systém a komunikace s podřízenými systémy

## (struktura a programová realizace řídicího systému a komunikace s programovatelnými automaty)

Návrh řídicího systému a komunikace s podřízenými systémy výrobní linky je nemyslitelný bez předchozí konkrétní definice požadavků na funkci a vlastnosti řídicího systému. Při vývoji podobně komplexních systémů nemusí vždy nutně první verze realizovaného systému splňovat všechny požadavky jakési maximální varianty, přesto však je nutné řešení provést takovým způsobem, že doplnění o složitější funkce a vlastnosti je relativně jednoduché a nevyžaduje zásadní změny v konceptu řešení.

Požadavky na řídicí systém definované v 4.2.1 obsahují jak požadavky na první jednodušší verzi systému, realizovanou v rámci této práce, tak i požadavky, které musejí být splněny v budoucí verzi systému, určené pro ostrý provoz.

Konkrétní popis mého řešení je rozdělen do dvou logických celků - 4.2 popisující řešení především z pohledu řídicího počítače a 4.3 z pohledu programovatelných automatů.

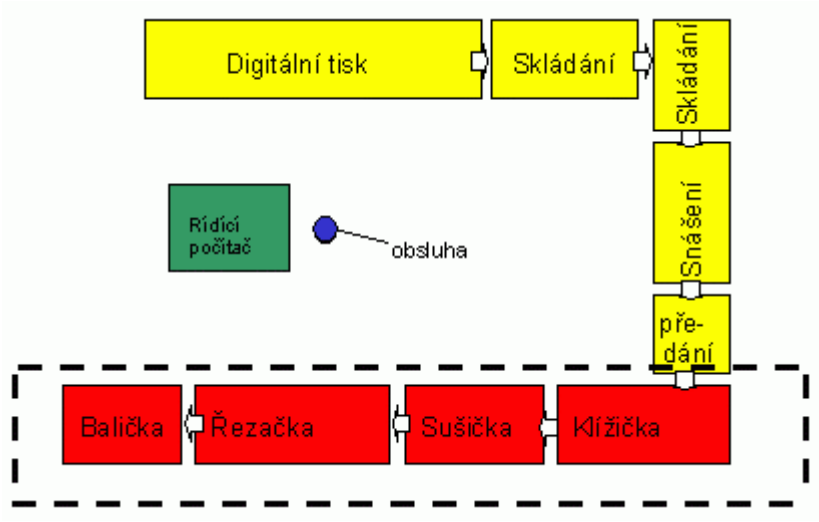
Všechna tato řešení byla v rámci této práce vyzkoušena jako reálně funkční a představují základ pro nově vyvíjený nadřazený řídicí systém výrobní linky firmy SRE.

## *4.1 Konkrétní konfigurace linky*

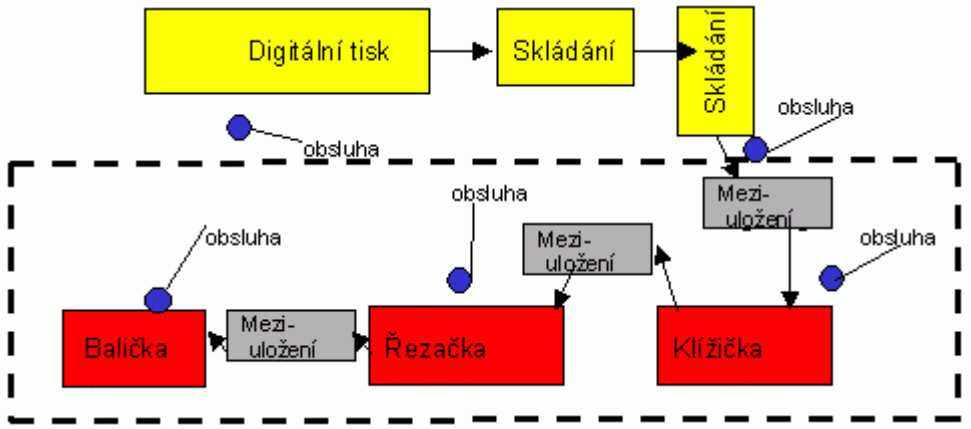
Dále popisované řešení je univerzální z hlediska konfigurace linky v tom smyslu, že je možno velice jednoduše definovat konfiguraci linky pomocí konfiguračního souboru a řídicí systém se pak automaticky přizpůsobí.

Ověření funkčnosti řešení jsem provedl na zcela konkrétní konfiguraci linky pro výrobu měkké lepené vazby. Schématicky je tato konfigurace znázorněna na obr. 4.1-1. V první fázi vývoje patří do této konfigurace pouze červeně znázorněné stroje v oblasti ohraničené čárkovanou čarou.

Pro porovnání je na obr. 4.1-2 znázorněno původní schéma výroby a rozložení strojů, které vyžadují meziukládání produktů. Je patrné, že tato konfigurace vyžaduje více pracovníků obsluhy strojů.




*Obr. 4.1-1 Schéma nové in-line linky*




*Obr. 4.1-2 Schéma výroby s meziukládáním produktů*

## *4.2 Řídicí systém*

Pod pojmem řídicí systém se dále rozumí komplexní soubor hardwarových prostředků a softwarových produktů, umožňující splnit dále uvedené požadavky.

### 4.2.1 Požadavky na řídicí systém

Na základě konzultací s pracovníky firmy SRE jsem definoval dále uvedené požadavky na funkci a vlastnosti řídicího systému. Ty obsahují všechny požadavky, které by měl systém v ideálním případě v budoucnu splňovat.

Jelikož jejich obsáhlost a komplexnost by v první fázi přesahovala nejen možnosti této práce, jsou ty požadavky, určené až pro druhou fázi vývoje uvedeny v závorce. Požadavky, které jsou relevantní až po úspěšném zakončení druhé fáze jsou pak uvedeny v závorce s hvězdičkou.

Všechny ostatní požadavky ale patří do první fáze a výsledkem této práce plně funkční řešení splňující tyto nároky.

Nadřazený řídicí počítač tedy musí být schopen provádět následující činnosti rovněž jako mít následující vlastnosti:

Vizualizace a řízení strojů linky

- řízení jednotlivých strojů ve výrobní lince
- přenos dat (parametrů procesu) na jednotlivé stroje, sledování počtu vyrobených produktů v rámci zakázky.

- grafické znázornění probíhající výroby knihy, stavu jednotlivých strojů, …
- (v případě výskytu chyby provedení příslušných opatření zabraňujících zastavení linky, posléze automatické napravení[1] chyby)
- (správa chybových hlášení s odpovídající vizualizací)
- (*správa plánovaných údržbových prohlídek jednotlivých strojů)
- (*pro budoucí rozšíření systému možnost konfigurace řídicího systému linky ve smyslu přidání či odebrání jednotlivých strojů, do určité míry přímo uživatelem)

Vstupní a výstupní data

Pro výměnu vstupních a výstupních dat jednotlivých zakázek, obsahující parametry specifikující výrobu knihy, počet výtisků, čas počátku výroby, dokončení zakázky a další, je použit soubor formátu JDF. Proto musí být řídicí počítač schopen provádět následující operace s JDF soubory:

- zadávání dat jak přímo z operátorského pracoviště (specifikace parametrů a vytvoření JDF souboru) tak i pomocí otevření externího JDF souboru
- načtení jednotlivých parametrů, jejich znázornění na operátorském pracovišti rovněž jako jejich editace a následně uložení do daného JDF souboru,
- načtení parametrů do vnitřních proměnných řídicího systému, po úspěšném zpracování zakázky jejich opětovné zapsání do výstupního JDF souboru,
- dle požadavků řízení správy zakázek manipulace s JDF soubory

Správa zakázek

Správa zakázek je jádrem celého řídicího systému. Mezi základní funkce musí patřit následující:

- Na základě daného JDF souboru, který specifikuje danou zakázku, načíst potřebná data pro výrobu knihy.
- Po dokončení zakázky pak následně tento soubor aktualizovat (například v případě odchylek rozměrů vyrobené knihy tyto změny zapsat do JDF souboru).
- Jednotlivé zakázky a jejich nejdůležitější údaje (například čas kdy má být zakázka hotová, start výroby zakázky, počet knih, jméno zákazníka a jméno zakázky) přehledně znázorňovat v tabulce, s možností přímo měnit tyto důležité údaje, jako například čas dokončení zakázky.
- Zobrazovat zakázky, které čekají na zpracování, právě probíhající výrobu a také historii již zpracovaných zakázek.
- Schopnost napojení na webovou aplikaci a její databázi, kde jsou obsaženy zakázky, které mají být vyhotoveny. To tedy znamená schopnost načíst příslušný záznam z tabulky této databáze, rovněž jako přístup k JDF souboru obsahující parametry zakázky.
- (* napojení na MES, MRP/ERP systém zákazníka)

[1] To znamená například v okamžiku, kdy dojde k zaseknutí knihy v některém ze strojů. V ideálním případě nedojde k zastavení linky, ale přesměrování produktů před nefuknčním strojem do zásobníku, po napravení chyby pak tento stroj s maximální možnou rychlostí zpracuje zakázky ze zásobníku a poté dojde k jeho synchronizaci s ostatními stroji v lince.

### 4.2.2 Vizualizační a řídicí software

Prvořadým kritériem pro výběr vhodného vizualizačního a řídicího software byla otázka možnosti a způsobu komunikace s programovatelnými automaty B&R. Jako nejschůdnější řešení se jevilo řešení založené na standardu OPC[1]. Toto řešení tak vyžaduje operační systém Windows. Vizualizační a řídicí software nekomunikuje přímo s automaty B&R, ale využívá takzvaného OPC serveru (v tomto případě B&R PVI OPC Serveru).

Z mnoha dostupných produktů na trhu, umožňujících vizualizaci a řízení procesu, rovněž jako napojení na OPC server, se jevil jako nejvhodnější programový produkt **Genesis32** firmy Iconics.

A to z následujících důvodů:

- plná verze vývojového prostředí zdarma, umožňující rovněž provoz v runtime režimu (bez zakoupení runtime licence však pouze vždy dvě hodiny vcelku)
- možnost napojení na libovolný OPC server a tím pádem i možnost komunikace s programovatelnými automaty B&R přes B&R PVI OPC Server
- dřívější zastoupení a přímá podpora produktů Genesis firmou B&R a tedy tím téměř zaručená bezproblémovost zmíněné komunikace mezi programovatelnými automaty a produkty řady Genesis (kompatibilita s B&R PVI OPC Serverem)

#### *4.2.2.1 Genesis32*

Pro vysvětlení dale používaných názvů jednotlivých produktů a komponent je jistě účelné uvést stručný přehled jednotlivých vývojových nástrojů z systému Genesis od firmy Iconics.

*Americká společnost ICONICS se od roku 1986 věnuje výhradně vývoji HMI/SCADA softwaru pro průmyslovou automatizaci pro operační systémy Microsoft Windows. V současné době nabízí řešení v podobě třetí generace svého SCADA sytému - GENESIS32, který je postaven na obecných standardech poskytující otevřenost systému, jeho modifikovatelnost a škálovatelnost. Při aplikaci těchto přístupů je systém adaptabilní pro další články kompletní systémové automatizace, například softwarového řízení, či sdílení informací s nadřazenými informačními systémy typu SAP/3.*

**GENESIS32** 7.0 představuje kompletní HMI/SCADA systém pro Win95/98/NT/2000/ME/XP. Systém obsahuje tři základní moduly pracující se standardy VBA, OPC a ActiveX.

4.2.2.1.1 Základní moduly

***GraphWorX32***: profesionální, výkonný grafický editor s knihovnou symbolů, možností importu bitmap a metasouborů a ActiveX kontrolů. Podpora vrstev podobně jako v AutoCadu. Další vlastnosti modulu: OPC veličiny, aliasy, výrazy a matematické formule, šablony obrazovek, gradientní výplně, podpora Drag&Drop v konfiguračním i runtime režimu. Splňuje OPC DA 2.x specifikaci. Integrován VBA 6.3 pro uživatelské skripty.

***TrendWorX32***: zobrazení grafů typu časového, logaritmického, XY, sloupcového a kruhového zapisovače s možností analýzy dat reálných a historických s použitím ActiveX prohlížeče. Historická data jsou ukládána do paměti nebo do databází MS Access, MSDE, MS SQL server 7.0, ORACLE 8.0. Nad historickými daty je možno generovat uživatelské zprávy do výše uvedených databází, MS Excel, HTML, nebo E-mail. Modul komunikuje s OPC DA 1.x, 2.x a OPC HDA 1.x servery. Integrováno VBA 6.3 pro uživatelské skripty.

[1] OPC = OLE for Process Control, OLE = Object Linking and Embedding. OPC je světový standard v současnosti založený na technologii COM a tedy zatím dostupný pouze pod operačními systémy Windows. Podrobněji viz 4.2.4.2

***AlarmWorX32*** – správa alarmů a událostí reálných (v reálném čase) a historických. ActiveX prohlížeč aktuálních alarmů a modul pro analýzu alarmů historických. Plná podpora OPC A&E, např. filtrování podle závažnosti, oblasti vzniku alarmů, distribuované řešení na bázi klient/server. Multimediální správa alarmů – řečový generátor, fax, telefon, textové pole, pop-up, E-mail, SMS, Video, MS Instant Messaging. Splňuje OPC A&E 1.x specifikaci. Integrováno VBA 6.3 pro uživatelské skripty.

4.2.2.1.2 Doplňkové moduly

Systém Genesis32 nabízí další doplňkové komponenty, takzvané „Tools“.

***DataWorX32*** – redundance a agregace dat, přemostění OPC serverů, globální veličiny, přístup přes Visual Basic.

***ControlWorX32*** – SoftPLC založené na IEC-61131 se vzorkovací frekvencí 1ms. Podpora jazyků: SFC, IL, FBD, LD, ST

***ScriptWorX32*** – prostředí pro správu paralelního běhu uživatelských skriptů ve VBA 6.3. Skripty je možno spouštět nezávisle na sobě jednorázově/ručně, periodicky, v závislosti na procesní OPC veličině, či alarmu a události.

***WebHMI*** – zpřístupnění GENESIS pomocí Internet Explorer nebo Netscape klientů. Bez jakékoliv instalace je možno zpřístupnit grafické, trendové a alarmové informace. Oboustranná komunikace se zabezpečením.

***ActiveX ToolBox*** – knihovna průmyslových ActiveX kontrolů komunikující přes OPC servery: měřící přístroj, posuvník, tlačítko, nádrž, časovač, událost, periodický, alarmový, atd.

***ActiveX ToolWorX*** – vývojový prostředek pro tvorbu OPC ActiveX prvků, komunikující přes OPC. Vývoj ve Visual C++ 6.0 s pomocí průvodců a grafického editoru GraphWorX32. Podpora prohledávání OPC serveru – Universální prohlížeč OPC tagů

***OPC ToolWorX*** – vývojový prostředek pro tvorbu OPC serverů/klientů. Podporuje OPC DA 2.x a A&E 1.x specifikaci, WinCE/95/98/NT/2000. Vývoj ve Visual C++ 6.0 s pomocí průvodce. Dodáván s Modbus OPC serverem ve zdrojové podobě.

***OPC Server*** – OPC server dle typu komunikačního protokolu.

***Dr.DCOM*** - Utilita usnadňující analýzu nastavení DCOM s možností autokonfigurace pro síťová řešení.

#### *4.2.2.2 Moduly Genesis použité pro řídicí systém*

Především z důvodů ceny runtime licence jednotlivých modulů je navržené řešení založené pouze na modulu GraphWorX. Díky tomuto modulu je možné jednak realizovat vizualizaci řízeného procesu, tak i pomocí VBA skriptů provádět vlastní řízení výrobního procesu.

### 4.2.3 Schéma komunikace a struktura řídicího systému

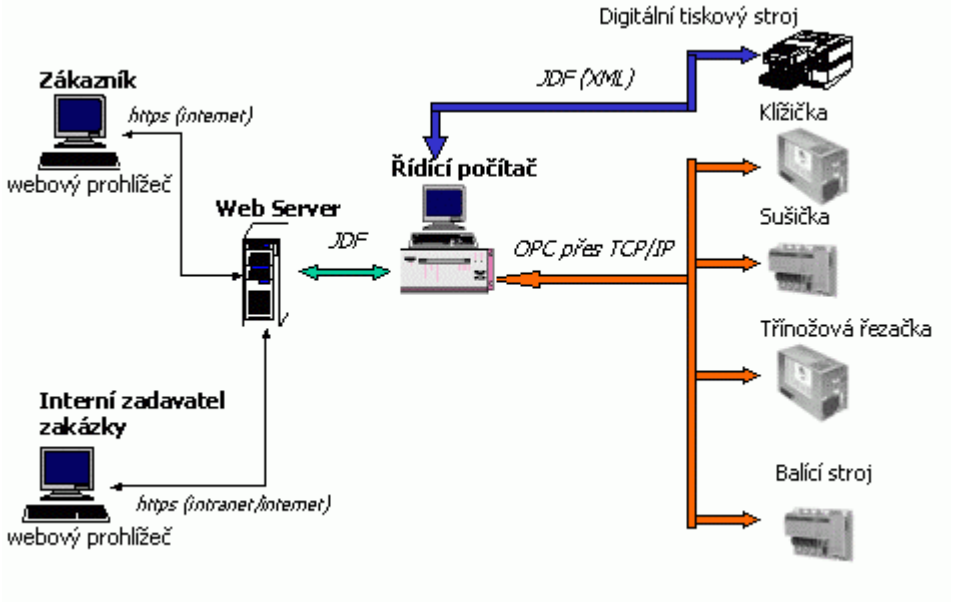
Komunikace v rámci výrobního procesu, struktura řídicího systému a způsob zadávání zakázky spolu úzce souvisí. Pro přehlednost jsou tyto části rozděleny do tří částí - 4.2.3.1 popisující možné způsoby komunikace se zaměřením na zvolené řešení, 4.2.3.2 zabývající se detailně způsobem zadávání zakázek a 4.2.3.3 obsahující rozbor struktury navrženého řídicího systému.

#### *4.2.3.1 Komunikace*

Na obr. 4.2-1 je znázorněno obecné schéma komunikace v rámci celého výrobního procesu knihy na navržené výrobní lince a to od zadávání zakázky v podobě specifikace parametrů výroby knihy, až po odesílání souborů dat jednotlivým strojům zařazených ve výrobní lince.

Na obr. 4.2-1 je zobrazena jedna z mnoha variant konfigurace výrobní linky. Tato konfigurace se může lišit jak z hlediska typů jednotlivých strojů a tedy tím i schopností vyrobit určitou škálu typů knihy (tuhá vazba, měkká vazba, …), ale také z hlediska způsobu komunikace řídicího počítače s jednotlivými stroji.

Kromě znázorněné komunikace pomocí OPC serveru, což znamená přímý přístup k konkrétním proměnným v jednotlivých strojích, je představitelná komunikace přímo pomocí JDF souboru. To však představuje poněkud vyšší nároky na programovatelné automaty strojů v lince. Tato práce je zaměřena především na variantu kdy všechny stroje komunikují s řídicím počítačem pomocí OPC proměnných. Možné je ovšem i takové nasazení jednotlivých strojů linky, kdy dodávka pro zákazníka – knihárnu - nebude obsahovat kompletní linku, ale pouze jednotlivé stroje bez řídicího počítače, které budou zařazeny do již stávajícího řetězce výrobní linky s cizím řídicím systémem. Vzhledem k současné hospodářské situaci je tato možnost velmi pravděpodobná a musí být tedy vyřešena, není ale součástí této práce.




*Obr. 4.2-1 Obecné schéma komunikace v rámci výrobního procesu*

V následující části jsou uvedeny různé možnosti zadávání zakázky.

#### *4.2.3.2 Zadávání zakázky (specifikace parametrů knihy spolu s údaji o zakázce)*

Pro zadávání zakázky jsem navrhl a implementoval v konečném řešení několik možných způsobů, které jsou naznačeny také na obr. 4.2-1.

Specifikace a objednání knihy koncovým zákazníkem

První možností, z hlediska dříve popisovaného konceptu výroby knihy na zakázku (Print on Demand, respektive Book on Demand, viz část 2.4.2) nejdůležitější, je specifikace a objednání knihy přímo koncovým zákazníkem pomocí standardního webového prohlížeče a to včetně vlastního obsahu knihy.

Na tomto místě je ale dlužno podotknout, že v rámci uváděného řešení není tento způsob zcela připraven pro komerční použití, protože v době vypracování této práce nebylo možno zahrnout do konceptu výrobní linky přímo rovněž digitální tiskový stroj. Především z toho důvodu, že kompletní implementace digitálního tisku do výrobní linky by dalece přesahovala časové možnosti dostupné pro vypracování této práce, nehledě na to, že nebylo možné mít k dispozici alespoň základní podklady pro tento druh stroje.

Na druhou stranu ale rozšíření uvedeného řešení o specifikaci vlastního obsahu knihy nebude vyžadovat zásadní změny v koncepci ani programové realizaci.

Specifikace rozměrů knihy zákazníkem (vydavatelem) či vlastním zaměstnancem knihárny

Tato možnost představuje výrazné zjednodušení doposud běžných postupů specifikace zakázky výroby knihy pro zákazníka knihárny – vydavatele. Vzhledem k malé míře standardizace výrobních postupů, rozměrů a typů knih je mnohdy relativně složité dohodnout takovou výslednou podobu knihy, která by vyhovovala jednak představám zákazníka[1], rovněž jako možnosti realizace na daných konkrétních strojích té které linky za přijatelnou cenu pro obě strany. Není tedy výjimkou zdlouhavý iterační postup, směřující ke kompromisu mezi spokojeností zákazníka a realizovatelností zakázky u výrobce (knihárny).

Díky existenci webového rozhraní pro specifikaci knihy, které plně reflektuje výrobní možnosti daného výrobce – průmyslové knihárny, dochází ke snížení počtu iteračních cyklů (v ideálním případě na jeden) a tedy i zefektivnění procesu specifikace knihy.

Jak je znázorněno na obrázku obr. 4.2-1, webové rozhraní je myslitelné nejen pro použití nakladatelem, ale i například pro vlastní zaměstnance knihařství pro vytváření zakázek v rámci konzultace se zákazníkem. Ale na rozdíl od původního postupu nemusí tito zaměstnanci znát natolik podrobně výrobní možnosti, protože kontrolu za ně provede webová aplikace.

Přímé zadání JDF souboru

Tato varianta je myšlena především pro budoucí spolupráci s digitálními tiskařskými stroji, které nemusejí komunikovat (a pravděpodobně nebudou) přímo s řídicím počítačem na obr. 4.2-1 rovněž znázorněným způsobem pomocí OPC Serveru. Tiskový stroj pošle synchronně s právě vytištěnou zakázkou JDF soubor, který kromě informací o tisku (tedy rozložení a obsahu stran), obsahuje rovněž informace pro svázání knihy.

Touto možností je rovněž umožněno přímé zadání specifikace knihy z operátorského pracoviště.

[1] Představy jsou mnohdy dílem kreativních designérů, kteří ale nemají příliš přesnou představu o technologických výrobních procesech a jejich omezeních.

#### *4.2.3.3 Struktura řídicího systému*

Na obr. 4.2-2 je schématicky znázorněna struktura použitých programů a jejich provázanost v rámci řídicího systému. Ten obsahuje dvě hlavní podskupiny, barevně odlišené a oddělené tlustou čarou.

Pravá podskupina představuje jádro řídicího počítače. Z hlediska operátora je viditelný grafický výstup z prostředí **GraphWorX**. To umožňuje interaktivně provádět požadavky operátora a představuje tak rozhraní mezi operátorem a dalšími podsystémy – programy.

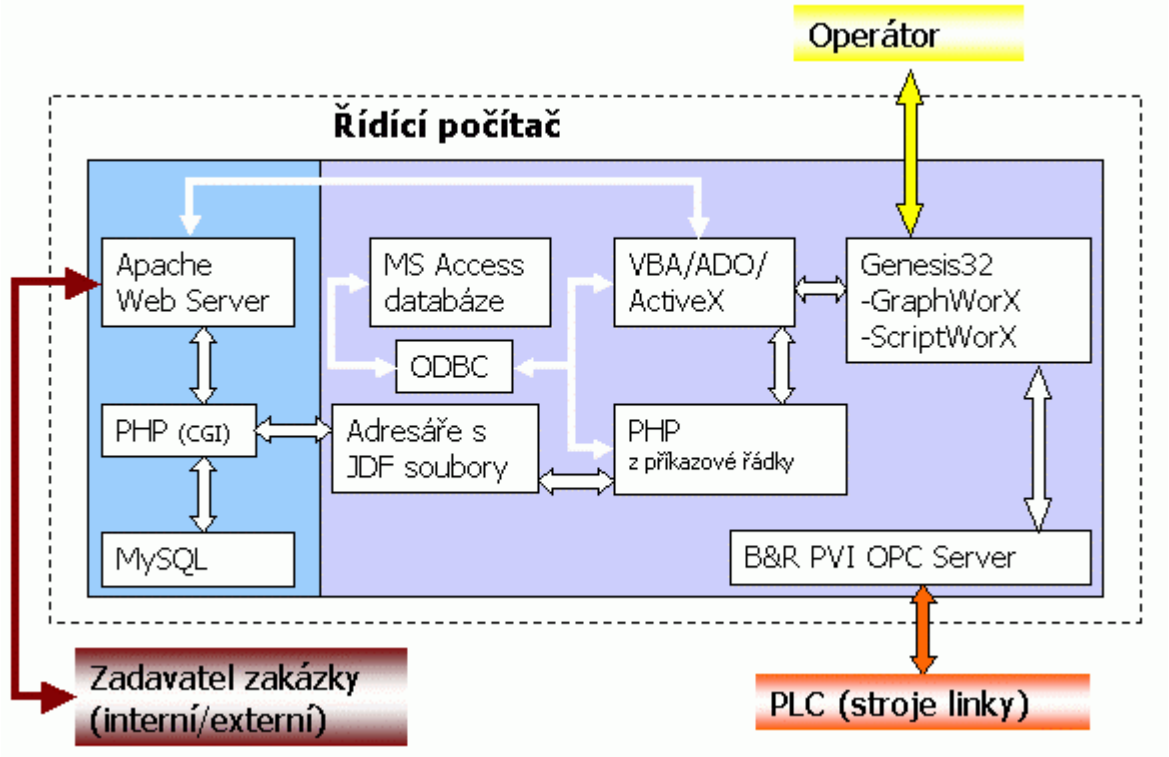
Moduly řídicího systému GraphWorX využívají prostředí jazyka Visual Basic for Applications (**VBA**). Ve VBA je realizována většina programů sloužících k řízení linky ze systémového hlediska. Ve VBA vytvořené skripty využívají technologie ActiveX Data Objects (**ADO**) spolu s ODBC rozhraním pro propojení s databází MS Access a prvků **ActiveX** pro propojení s Webovým serverem. Programy rovněž spouštějí pomocí funkce shell, umožňující provádět příkazy jako z příkazové řádky, skripty napsané v jazyce PHP a to spuštěním PHP právě z příkazové řádky. Podrobnější rozbor řešení využívající GraphWorX, VBA, ADO a ActiveX je možno nalézt v části 4.2.4.

Spuštění **PHP** z příkazové řádky je jedna z možností jak provést skripty napsané v PHP. Tuto možnost jsem s výhodou využil například pro čtení JDF (XML) souborů. Ty byly napsány takovým způsobem, že je možné použít pro čtení parametrů knihy obsažených v JDF souboru stejný skript jak v případě webové aplikace tak řídicího počítače. To samozřejmě výrazně zjednodušuje případné budoucí modifikace daného skriptu, protože vše je nutné měnit pouze jednou (na jednom místě).

Databáze MS **Access** je použita pro správu zakázek. Jsou v ní definovány čtyři tabulky – jedna pro přípravu zakázek, druhá pro vlastní zakázky určené k výrobě, třetí pro právě vyráběné zakázky a čtvrtá sloužící ke sledování historie vyrobených zakázek. Pro aktualizaci tabulky pro přípravu zakázek je rovněž využit skript v PHP, který využívá rovněž ODBC rozhraní. Skript testuje adresář s JDF soubory a v případě, že se vyskytne nový soubor (vložený buď z webové aplikace po odeslání vytvořeného JDF souboru zákazníkem k výrobě, nebo přímo z operátorského pracoviště) přečte z něj informace o zakázce a uloží jej do tabulky pro přípravu zakázek. Postup zpracování zakázky řídicím počítačem je podrobněji vysvětlen v části 4.2.4.1.

Rozhraní mezi programovatelnými automaty – řídicími jednotkami jednotlivých strojů linky – představuje pak **B&R PVI OPC Server**. Z modulů Genesis je možné přistupovat k předem definovaným OPC proměnným libovolného OPC serveru. Vzhledem k tomu, že programovatelné automaty strojů linky jsou od firmy B&R, je použit B&R PVI OPC Server. Ten přes protokol TCP/IP umožňuje zapisovat či číst hodnoty definovaných proměnných v jednotlivých automatech. Další informace k OPC serveru a OPC komunikaci vůbec jsou uvedeny v části 4.2.4.2.

Levá podskupina je určená především jako rozhraní pro „vnější“ komunikaci s zákazníkem. Programy této podskupiny můžou být spuštěny teoreticky přímo na fyzicky stejném počítači jako řídicí systém, což přináší úspory z hlediska nákladů na potřebný hardware, na druhou stranu ale zvyšuje bezpečnostní rizika. Ústředním modulem je webový server (zde konkrétně **Apache**, ale teoreticky libovolný server podporující PHP). Webový server pak spouští skripty v jazyce **PHP**, které jako databázi využívají **MySQL**.




*Obr. 4.2-2 Struktura programů a jejich provázanost v rámci řídicího počítače*

### 4.2.4 Programová realizace

V další části je uveden popis programové realizace řídicího počítače. Navržené řešení je univerzální z hlediska konfigurace linky. To znamená, že je možno konfigurovat počet strojů v lince stejně tak jako spektrum procesů, které mají být na strojích prováděny. Konkrétní popis je pak uveden na příkladu konfigurace linky sestávající se ze tří strojů – lepícího a vázacího stroje, ořezávacího stroje a balícího stroje.

Nejdříve je obecně popsán systém pro řízení a sledování zakázek (část 4.2.4.1).

V části 4.2.4.2 je vysvětlena komunikace pomocí standardu OPC obecně, stejně tak jako konkrétní použití v rámci GraphWorX.

Navržené řešení je založeno na objektově orientovaném přístupu a proto jsou ve zvláštní části 4.2.4.3 popsány vytvořené třídy a struktura použití jejich instancí (objektů).

Jak již bylo výše uvedeno, jako vizualizační a řídicí software jsem vybral produkt Genesis - GraphWorX. V tomto prostředí je tedy realizována vlastní vizualizace a řízení procesu, což je popsáno v části 4.2.4.4.

Podrobnější popis vlastního řízení strojů je možno nalézt v části 4.2.4.5. V této části je popsáno, jakým způsobem je realizována možnost dynamické konfigurace strojů linky na základě konfiguračního souboru, dle jakého funkčního diagramu je řízen výrobní proces a uveden modelový příklad výroby zakázek. To vše v návaznosti na hlavní použitý programovací jazyk VBA.

#### *4.2.4.1 Systém řízení a sledování zakázek*

Je zřejmé, že pro realizaci správy zakázek řídicím systémem je vhodné využít možností databázového systému. Jako nejvýhodnější se jevila databáze Microsoft Access a to z následujících důvodů.:

- V případě použitého operačního systému Windows je dostupný (zdarma) ovladač pro soubory databáze MS Access, umožňující přístup přes ODBC rozhraní k této databázi.
- Jednoduchý přístup z VBA přes ADO rozhraní
- Snadná tvorba struktury databáze pomocí programu MS Access ve fázi vývoje.

- Toto řešení ale nevyžaduje instalovaný program MS Access jako takový na řídicím počítači (a tedy nevznikají žádné další náklady). Přestože toto řešení není rozhodně úplně nejlepší z hlediska spolehlivosti, je pro tyto účely vyhovující. Použití přímo programu MS Access by bylo jistě nevhodné pro průmyslové účely. Na druhou stranu přístup přes ODBC rozhraní k databázi MS Access obsahující méně než 10 000 záznamů je pro malé aplikace tohoto typu z hlediska spolehlivosti dostačující.

Pro řízení a sledování zakázek jsem navrhl řešení založené na čtyřech tabulkách, které slouží ke správě zakázek od okamžiku kdy zákazník odeslal zakázku k výrobě až po její dokončení.

- tabulka „jobs_prepared“
- tabulka „jobs_to_do“
- tabulka „jobs_in_progress“
- tabulka „jobs_finished

Všechny tabulky mají identickou následující strukturu

| *Jméno pole* | *Typ pole* | *Velikost* | *prim. klíč* | *požadované* |
|---|---|---|---|---|
| file_name | text | 40 | x | yes |
| orig_file_name | text | 40 | | |
| customer_name | text | 40 | | |
| customer_id | text | 20 | | |
| customer_job_name | text | 50 | | |
| start_time | text | 25 | | |
| end_time | text | 25 | | yes |

Tabulka jobs_prepared

Tabulka „*jobs_prepared*“ je určena pro přípravu zakázek určených k výrobě. Slouží jako rozhraní pro zadávání zakázek/JDF souborů do systému. Principielně existují tři způsoby, jakým se může JDF soubor přidat do této tabulky:

- automaticky z webové aplikace v případě, že uživatel webového rozhraní odešle zakázku k výrobě
- manuálně z operátorského pracoviště
- (automaticky pokud jej pošlou na řídicí systém předřazené systémy – například digitální tiskový stroj)

Před přidáním JDF souboru do této tabulky proběhne kontrola, zda parametry zakázky pro výrobu knihy jsou zadány správně a zda odpovídají výrobním možnostem aktuální konfigurace výrobní linky. V kladném případě dojde k přidání, v záporném je pak znázorněno příslušné chybové hlášení.

Parametry zakázky v této tabulce je dále možné upravovat a to buď přímo nejdůležitější údaje obsažené v řádku záznamu tabulky, nebo přes dialogové okno rovněž parametry výroby knihy.

Tabulka jobs_to_do

Tato tabulka slouží jako fronta zakázek určených ke zpracování. Přidání zakázky do této tabulky je možné pouze z operátorského pracoviště přesunutím zakázky z tabulky jobs_prepared.

Zde není již možné měnit parametry výroby knihy, pouze změnou údaje, kdy má být zakázka hotová, je možné změnit pořadí zakázky (přesunout ji nahoru nebo dolů). Řídicí systém totiž řadí zakázky vzestupně podle doby dokončení a vždy první zakázka je v případě připravenosti výrobní linky na novou zakázku přesunuta do tabulky jobs_prepared a tím zahájena její výroba.

Tabulka jobs_in_progress

V této tabulce jsou uloženy právě vyráběné zakázky. Při počtu N strojů ve výrobní lince provádějících dokončovací operace může být totiž teoreticky právě aktivních N zakázek (a dále N+1 – tá zakázka čekající na výrobu)

V případě, že všechny operace na jednotlivých strojích byly úspěšně dokončeny, je zakázka hotová a je ji možné přesunout do tabulky jobs_finished.

Tabulka jobs_finished

Do této tabulky jsou přemístěny všechny úspěšně vyrobené zakázky. Ale na rozdíl od ostatních tabulek je zde před název zakázky (file_name) předřazen řetězec obsahující datum a čas ve formátu YYMMDDHHmm (význam jednotlivých písmen odpovídá běžnému způsobu zápisu data a času):

- YY – poslední dvojčíslí roku
- MM – měsíc
- DD – den
- HH – hodina
- mm – minuta

### *4.2.4.2 Komunikace přes OPC (server a klient)*

4.2.4.2.1 Úvod do OPC

OPC (OLE für Process Control) je průmyslový standard, který byl vyvinut v rámci spolupráce předních světových výrobců, působících v oblasti automatizace a hardwaru, a společností Microsoft. Organizace, která spravuje tento standard se nazývá OPC Foundation (www.opcfoundation.org).

*Obr. 4.2-3 Logo OPC*

Standard OPC je založen na technologiích Microsoftu OLE (Object Linking and Embedding) und COM (Component Object Model). Je složen ze souboru standardních rozhraní, vlastností a metod, které jsou používány v řízení procesů a při tvorbě automatizačních klientů. Technologie OLE/COM definují, jakým způsobem spolu jednotlivé softwarové komponenty spolupracují a vyměňují si informace.

OPC nabízí jednotné rozhraní pro komunikaci mezi rozdílnými zařízeními určených pro řízení procesů, a to nezávisle na řídicím softwaru, který je použit pro řízení procesů.

Z uživatelského hlediska jsou podstatné pouze dvě následující aplikace:

- OPC Server
- OPC Client (klient)

*OPC Server* představuje rozhraní mezi zařízením (či zařízeními) určeným pro řízení procesu (v našem případě programovatelnými automaty) a rozhraním OPC v rámci počítače, na kterém je OPC Server spuštěn, popřípadě dalších počítačů v síti na tento počítač napojených. Jednotlivé proměnné[1] definované v konfiguraci OPC Serveru jsou přístupné přes OPC rozhraní právě OPC klientům.

OPC klient je tedy aplikace, která umožňuje přístup k proměnným všech dostupných OPC Serverů, ať již fyzicky v rámci počítače, na kterém je spuštěna, či přes síť. konkrétně tedy například v případě vizualizačního a řídicího systému GraphWorX je automaticky spuštěna na pozadí a zprostředkovává čtení a zápis jednotlivých proměnných (anglicky nazývaných Tags[2]) definovaných v rámci vizualizace procesu.

4.2.4.2.2 B&R PVI OPC Server

B&R PVI OPC Server je OPC kompatibilní server, který díky použití rozhraní nazývaného B&R Process Visualization Interface (PVI), umožňuje komunikaci s různými I/O zařízeními a

[1] Hodnoty těchto proměnných jsou většinou cyklicky obnovovány v předem daném časovém intervalu, například 1000ms

[2] Typickou vlastností licencí současných vizualizačních softwarů je právě omezení počtu tagů, ke kterým je možno současně přistupovat. Čím větší počet tagů, tím více stojí verze licence. Nejmenší varianta licence začíná mezi 50-75 Tagy dle výrobce a je škálovatelná po určitých stupních (například 50, ale i 25 či méně, opět závislé na výrobci) až po nejdražší variantu neomezeného počtu tagů.

může tak zprostředkovat data OPC klientům. PVI je komponentou rozhraní B&R AUTOMATION NET.

PVI OPC Server podporuje OPC Data Access Version 1.0 a 2.0 stejně tak jako OPC Alarms & Events Version 1.0 a může být použit pomocí technologie DCOM pro Intranetové a Internetové aplikace.

Další vlastnosti PVI OPC Serveru jsou:

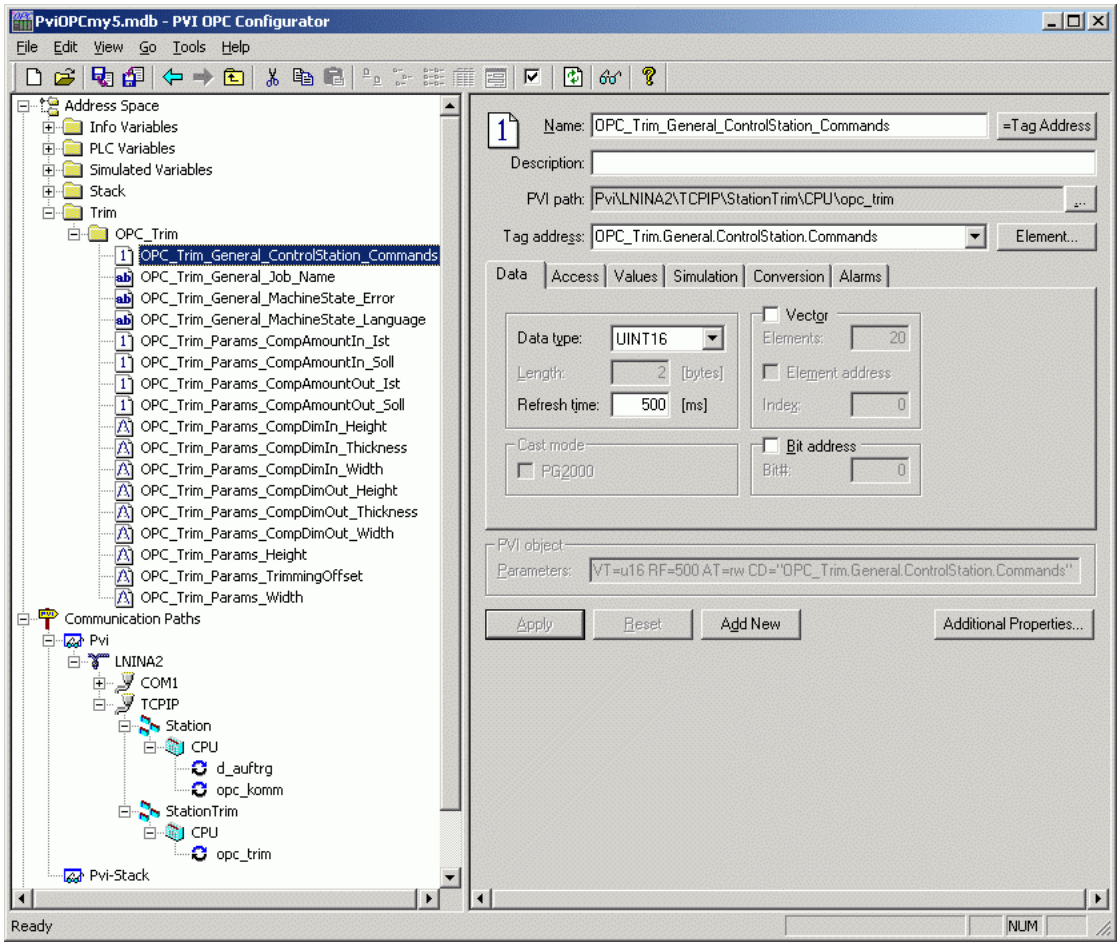
- uživatelské rozhraní podobné Windows Exploreru
- režim Monitor View pro zobrazení hodnot Tagů
- multiplikátor tagů pro rychlé vytvoření většího množství tagů
- import proměnných z vývojového prostředí Automation Studio (vývojové prostředí pro programování automatů B&R, viz také 3.2.2.2) či přímo online z programovatelného automatu (automatická definice proměnných)
- import standardních PVI konfiguračních souborů
- import/export do/z CSV (Comma Separated Value) formátu
- podpora více lokálních či vzdálených PVI spojení s automaty
- podpora logických skupin (adresářů) pro jednoduchou konfiguraci a správu tagů
- rozšířená možnost zjišťování kvality OPC dat a typová konverze
- interní simulační možnosti pro konfiguraci a testy
- výstup z PVI Data Logger hlášení pro hledání chyb
- konfigurace tagů OPC serveru je uložena v databázi MS Access, takže je teoreticky možné dále automatizovat správu konfigurace tagů

Komponenty používané v souvislosti s B&R PVI OPC Serverem

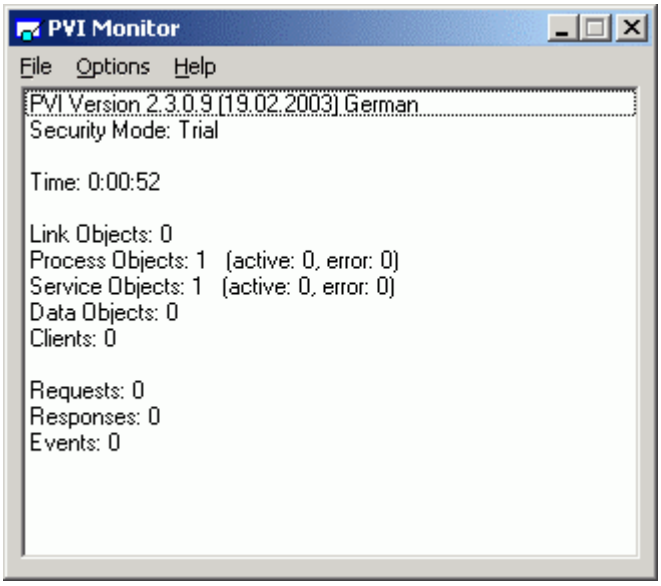
Vlastní aplikace serveru je z uživatelského hlediska neviditelná. Pro jeho správu jsou však k dispozici následující komponenty

- OPC Configurator – aplikace sloužící ke konfiguraci tagů. Ta je ústřední a musí ji uživatel nutně spustit, aby mohl definovat odkud a jaké proměnné mají být dostupné.
- OPC Sample Client – může sloužit především k testovacím účelům. Jak název napovídá, jedná se o klienta.
- OPC Registry Check
- PVI Monitor – monitoruje komunikaci, udává počet aktivních spojení. V případě provozu na jiném počítači než B&R IPC (hardwarově kontrolováno) aktivuje takzvaný „Trial Mode“, který umožňuje komunikaci pouze dvě hodiny vcelku. Po skončení této lhůty je nutné OPC Server pomocí PVI Manageru ukončit a znovu spustit.
- PVI Manager

Pro názornost je na obr. 4.2-4 znázorněno okno aplikace OPC Configurator, a na obr. 4.2-5 pak PVI Monitoru.




*Obr. 4.2-4 OPC Configurator*




*Obr. 4.2-5 PVI Monitor*

#### 4.2.4.2.3 OPC klient (přístup k OPC proměnným z GraphWorX)

V dále popisovaném navrženém řešení není OPC klient přímo viditelný. Pro přístup k OPC proměnným využívám možnosti prostředí GraphWorX a konkrétně objekty typu ProcessPoint. Pro každou proměnnou je nutné definovat objekt tohoto typu a specifikovat jeho DataSource – tedy zdroj dat (viz obr. 4.2-6).

Jako Data Source je v GraphWorX v zásadě obecně možné uvést následující typy:

- lokální proměnná
- globální proměnná
- OPC Tag
- výraz kombinující různé funkce a proměnné

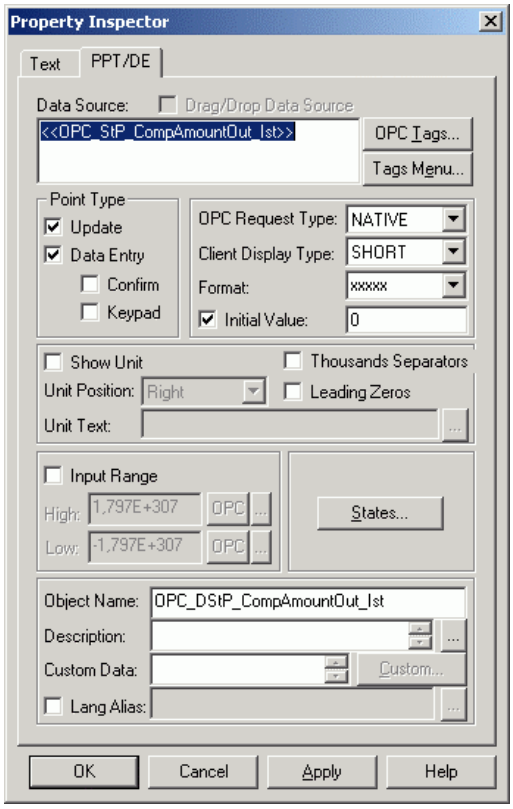
Všechny proměnné je možné definovat přímo. Například lokální proměnná jména pokus se napíše ve tvaru: `~~pokus~~`, pokud není ještě definována, rovnou se tím definuje; OPC proměnnou je možné pomocí menu OPC Tags… také definovat přímo – zápis pak vypadá:

```
B&R.PviOPC.2\Stack.OPC_Stack.
OPC_Stack_Params_CompAmountOut_Ist
```

S výhodou lze využít také takzvaných aliasů. Do položky DataSource se nenapíše celý název, ale pouze alias. V tomto případě tedy například:

```
<<OPC_StP_CompAmountOut_Ist>>
```

Stejně jako značky `"~~"` uvozují lokální proměnné, značky `"<<"` resp. `">>"` uvozují aliasy.




*Obr. 4.2-6 Vlastnosti objektu ProcessPoint*

Přiřazení skutečného zdroje dat je pak možno realizovat pomocí v GraphWorX dostupných funkcí v VBA buď z textové proměnné nebo ze souboru, obsahující definice v daném formátu, a kdykoli při runtime režimu

Příslušný řádek souboru pro uvedený alias by pak měl následující tvar:

```
OPC_StP_CompAmountOut_Ist     B&R.PviOPC.2\Stack.OPC_Stack.OPC_Stack_Params_CompAmountOut_Ist
```

Formát řádku tohoto souboru je tedy:

```
název_aliasu<tabulátor>zdroj
```

Pomocí procedur ve VBA je pak možno do těchto proměnných zapisovat či načítat jejich hodnoty.

#### 4.2.4.2.4 Definované proměnné pro zápis parametrů a řízení PA

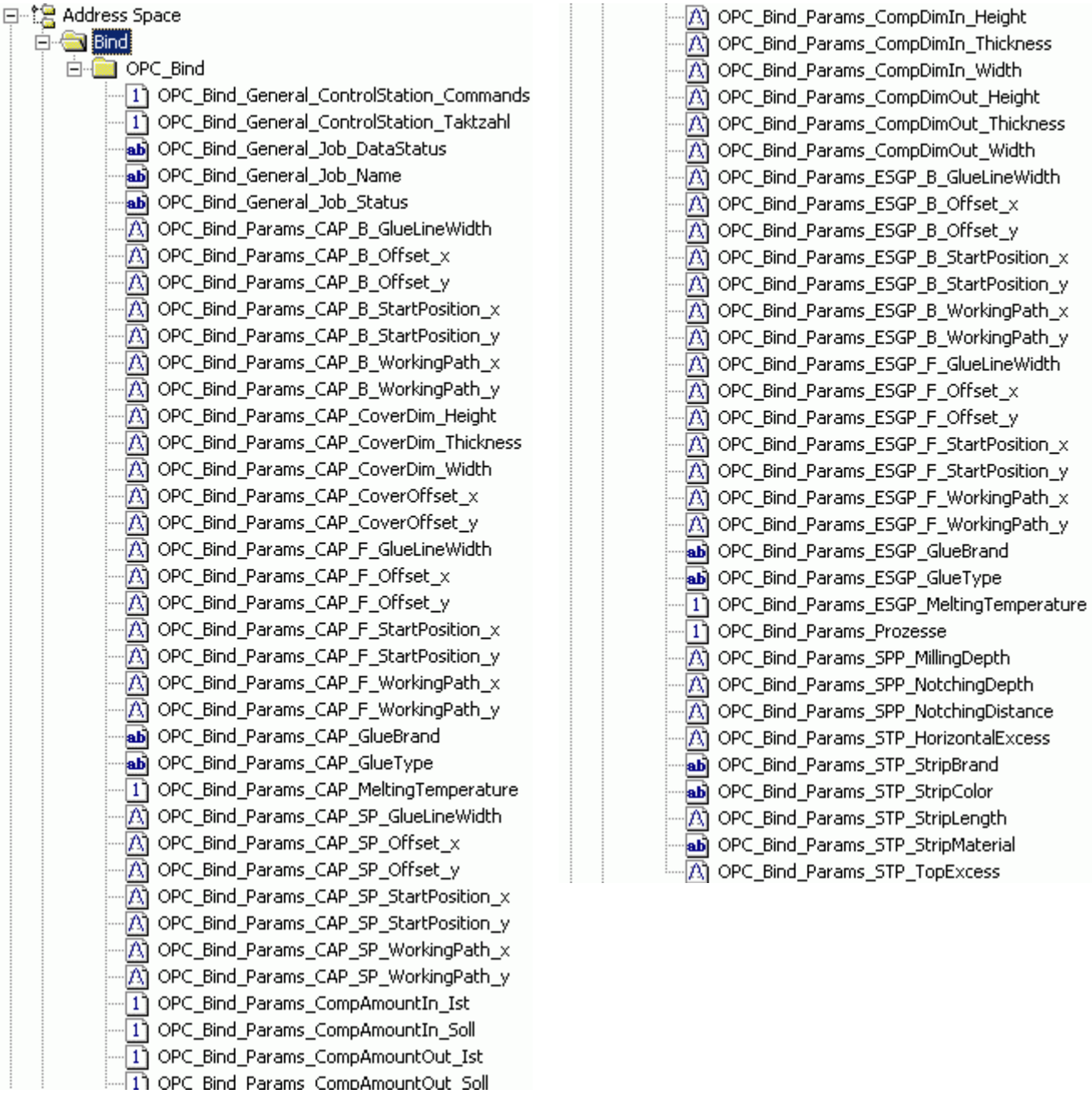
Pro realizaci komunikace řídicí system – programovatelné automaty pomocí OPC serveru je zřejmě nutné definovat proměnné, které budou sloužit pro zápis parametrů výroby knihy řídicím systémem do programovatelných automatů, stejně tak jako řídicí proměnné a rovněž další určené pro zpětné hlášení stavu jednotlivých strojů řídicímu systému.

Pro tento účel jsem (na základě JDF) definoval strukturu parametrů pro jednotlivé stroje v programovacím prostředí B&R Automation Studio (AS) a tyto proměnné pak importoval pomocí volby „import online“ do konfigurace programu OPC Configurator pro OPC server.

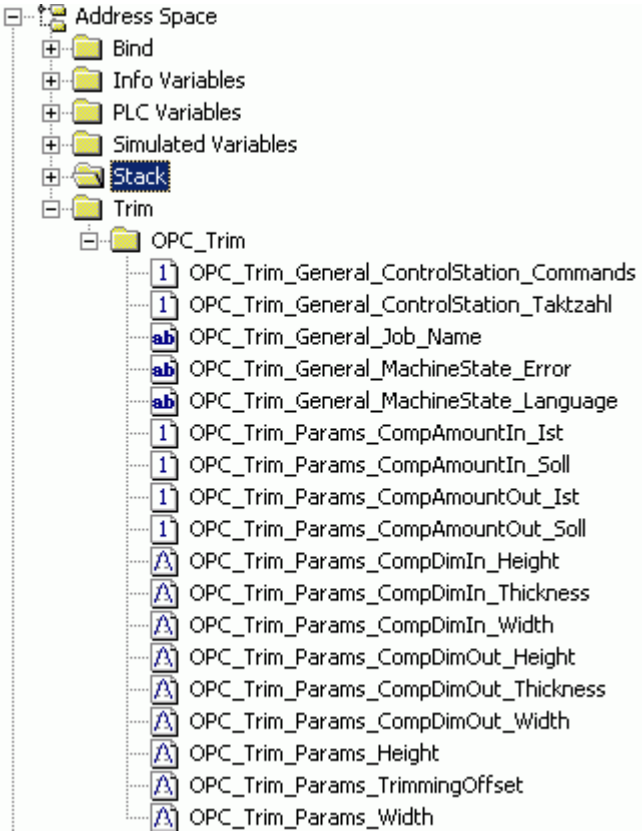
Struktura proměnných pro Automation Studio (pro programovatelné automaty) je dále popsána v části 4.3.1.

Na obr. 4.2-7 je struktura OPC proměnných pro stroj „Bind“ – lepící a vázací stroj, na obr. 4.2-8 pro stroj „Trim“ – řezačku a na obr. 4.2-9 pro „Trim“ – balící stroj.

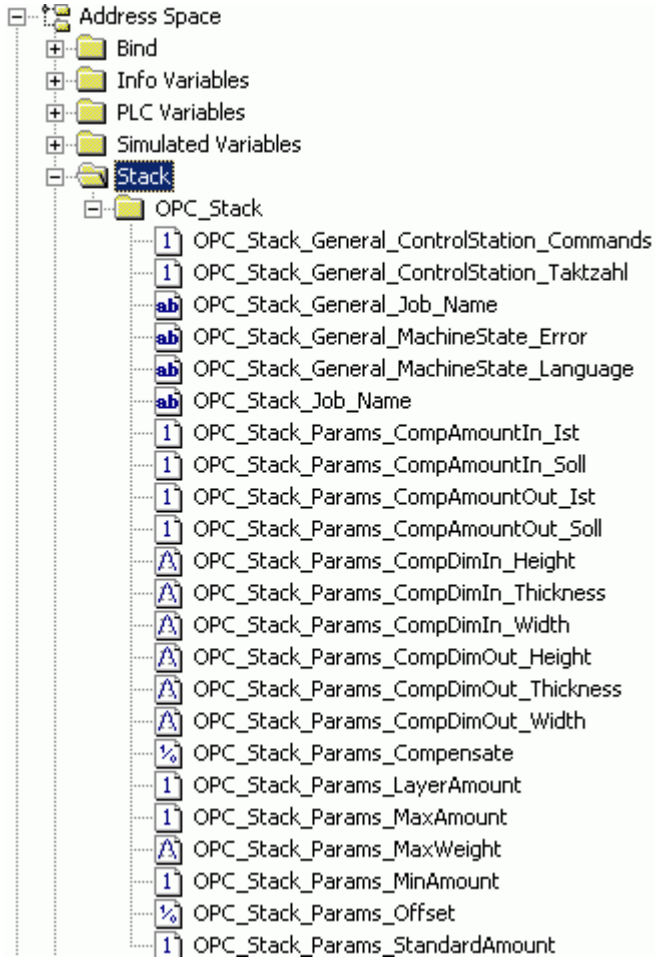
Na rozdíl od stromové struktury v AS (viz část 4.3.1) jsou zde všechny proměnné v podstatě na stejné úrovni, z jejich názvů (předpon) je však zřejmě původní stromová struktura odpovídající AS lehce dekódovatelná).




*Obr. 4.2-7 OPC proměnné pro stroj „Bind“*




*Obr. 4.2-8 OPC proměnné pro stroj „Trim“*




*Obr. 4.2-9 OPC proměnné pro stroj „Stack“*

#### *4.2.4.3 VBA - vytvořené třídy a struktura použití jejich instancí (objektů)*

Pro programovou realizaci řízení takto komplexního systému je zřejmě vhodné a téměř nevyhnutelné aplikovat alespoň základní postupy objektově orientovaného programování (OOP). Předmětem této práce není úvod do objektově orientovaného programování, přesto je ale účelné připomenout alespoň základní pojmy z této oblasti:

Objektově orientovaný přístup by měl doprovázet všechny fáze vývoje softwaru, které je možno rozdělit následovně:

- analýza problému
- definice rozhraní (mezi objekty)
- programování (implementace)
- dokumentace

Důležitou vlastností realizace dle OOP je právě oddělení specifikace (definice) od implementace, tzn. uzavření implementačních detailů do zvenku nepřístupných míst (ukrývání vnitřní informace).

Mezi základní pojmy OOP patří třídy, objekty, atributy (vlastnosti) a metody.

Třídy (Classes)

Třída popisuje skupinu objektů podobných vlastností. Nejedná se přímo o proměnnou, ale o typ proměnné, lépe řečeno typ objektu. *V této práci tedy například třída s názvem MachineClass.*

Objekty (instance)

Objekt je popisován jako instance jeho třídy. Je charakterizován svým jménem, vlastnostmi a metodami. Tedy například instance třídy MachineClass jsou prvky pole Machines, tedy objekty Machines(0), Machines(1), …, Machines(N)

Vlastnosti (atributy) – properties

Vlastnosti objektů jsou jejich lokální datové struktury uvedené v definici třídy. *Objekt Machines(0) má tedy například vlastnosti Name či MachineStatus.*

Metody (operace)

Metody charakterizují dynamické chování objektů tím, že definují funkce či procedury, které mohou být u objektů použity. Objekt Machines(0) má tedy například metodu WriteParamsToOPC, která zapíše parametry zakázky uložené v poli (vlastnosti) MachinesParamsArray do proměnných OPC Serveru.

Další neméně důležité pojmy OOP již jen ve stručnosti:

- zapouzdřednost - systém je tvořen z jednotlivých objektů. Pouze metody zabezpečují přístup k vlastnostem (lokálním datovým strukturám)
- dědičnost - objekty mohou dědit své vlastnosti, tzn. zejména strukturu lokálních dat a množinu svých metod, od jim nadřazených objektů. Vzniká tak hierarchie objektů, přičemž nejobecnější objekty jsou v nižších stupních hierarchie postupně upřesňovány (a tím také rozmělňovány) do mnoha konkrétnějších tříd.
- polymorfismus - je obecně vlastnost, díky níž může mít totéž jméno (proměnné, funkce, procedury,...) více rozdílných významů. V OOP to specielně znamená, že metody objektů, i když mají stejná jména, se chovají různě podle toho, v jakém kontextu jsou zavolány.
- (posílání zpráv - objekty mezi sebou komunikují posíláním zpráv)

#### 4.2.4.3.1 Definované třídy

Pro realizaci programu řídicího systému jsem navrhl třídy objektů zobrazené v tab. 4.2-1.

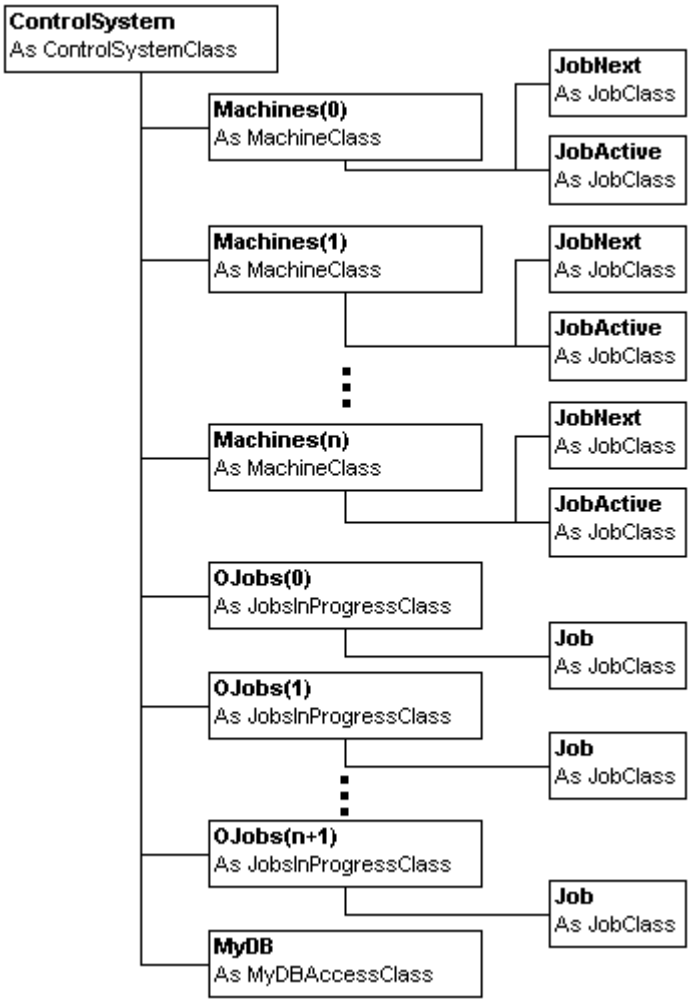
| **Název třídy** | **Stručný popis** |
|---|---|
| ControlSystemClass | Třída určená pro řízení jednotlivých strojů v lince |
| JobClass | Třída definuje objekt zakázky. Objekty tohoto typu jsou využívány pro uchování parametrů zakázky. |
| JobsInProgressClass | Parametry jednotlivých právě vyráběných zakázek spolu s informací o tom, v jaké fázi se zakázka na jednotlivých strojích nachází, jsou uchovány v poli typu této třídy. |
| MachineClass | Instance této třídy umožňují zobrazit konfiguraci výrobní linky – tedy definují jednotlivé stroje. Zároveň slouží jako rozhranní mezi objektem řídicím stroje (instance třídy ControlSystemClass) a OPC proměnnými, přes které dochází ke skutečnému řízení programovatelných automatů jednotlivých strojů. |
| MyDBAccessClass | Tato třída definuje operace pro přístup k tabulkám databáze zakázek a manipulace s jejich záznamy. Slouží rovněž pro s tím související manipulaci s JDF soubory, stejně tak jako pro jejich čtení a konverzi. |

*Tab. 4.2-1 Třídy definované pro řídicí systém*

#### 4.2.4.3.2 Struktura objektů vytvořených na základě definovaných tříd

Na obr. 4.2-10 je uvedena hierarchická struktura objektů (a jejich typů) použitých pro řízení průběhu výroby. Vzhledem k tomu, že počet strojů v lince může být obecně různý, je symbolicky znázorněno celé pole objektů Machines typu MachineClass, které má n prvků (n strojů). Pro

n strojů je pak definováno pole OJobs, které má n+1 prvků – zakázek (teoreticky n aktivních zakázek = právě vyráběných, n+1 – tá zakázka pak kompletně ve stavu waiting.




*Obr. 4.2-10 Struktura objektů vytvořených na základě definovaných tříd*

4.2.4.3.3 Proměnné a metody jednotlivých tříd

ControlSystemClass

*Proměnné (důležité):*

**Private MyDB As MyDBAccessClass**

**Private Machines(MACHINESMAX) As MachineClass**

- pole objektů třídy MachineClass

**Private OJobs(JOBSMAX) As JobsInProgressClass**

*Metody*

**Public Sub GlobalInit()**

- Inicializuje jednotlivé objekty této třídy. Tato procedura je volána po přepnutí projektu GraphWorX do režimu runtime. Dle konfiguračního souboru vytvoří objekty Machines dle konfiguračního souboru (vytvoří tak aktuální konfiguraci strojů linky spolu s definicí procesů, které jsou jednotlivé stroje schopné provádět).

**Public Sub Deinitialize()**

- zruší všechny objekty této třídy (uvolní je z paměti)

**Public Sub CyclicProductionControl()**

- Stěžejní procedura této třídy. Je cyklicky volána v hlavním programu.
- Dává povel ke startu zakázky pokud je první stroj v lince připraven na nový job (přesune zakázku z tabulky jobs_to_do do tabulky jobs_in_progress a zapíše parametry zakázky do prvního volného objektu pole OJobs.

- Volá proceduru ControlMachine pro všechny stroje linky
- Kontroluje, zda některá ze zakázek ve výrobě není již kompletně hotová (stav všech strojů FINISHED). V tom případě přesune zakázku do tabulky jobs_finished
- Dále voláním procedury DisplayJobStates zobrazuje cyklicky stav všech zakázek na displej.

**Public Sub ControlMachine(Machines() As MachineClass, MachineNr As Integer)**

- Procedura zodpovědná za řízení jednotlivých strojů. Založená na funkčním diagramu dle obr. 4.2-15. Pro řízení řetězce stavů využívá proměnné objektu Machines(MachineNr) (tedy konkrétního stroje) *ControlJobState*, definující ve kterém stavu stavového diagramu se právě nachází komunikace mezi řídicím systémem a programovatelným automatem.

**Private Sub DisplayMachinesConfiguration(NrOfMachines As Integer)**

- Zobrazuje aktuální konfiguraci strojů linky

**Private Sub SetJobObject(ObjectToRead As JobClass, ObjectToSet As JobClass)**

- pomocná procedura pro přiřazení objektů typu JobClass

**Private Function GetJobNr(MachineNr As Integer, WhichStatus As String) As Integer**

- Funkce vrátí číslo (pořadí) zakázky, u které jako první najde u stroje daného číslem MachineNr stav zakázky daný parametrem WhichStatus

**Private Sub SetOJob(OJobToSet As JobsInProgressClass, ByVal OJobToRead As JobsInProgressClass)**

- Pomocná procedura pro přiřazování objektů při vymazávání hotové zakázky z pole OJobs

**Private Sub ReadConfigFile(FileToRead As String, ConfigArray() As String)**

- Procedura vrátí konfigurační parametry v poli daném na místě ConfigArray ze souboru daném parametrem FileToRead

JobClass

Vlastnosti

**Public Name As String**

**Public CompAmountOut_Soll As Integer**

- proměnná sloužící k uchování počtu produktů, který má být v zakázce vyroben

**Private MachinesParamsArray() As String**

- pole parametrů zakázky

Metody

**Public Sub SetParamsArray(InParamsArray() As String)**

- Uloží parametry z pole InParamsArray do pole MachinesParamsArray

**Public Sub GetParamsArray(OutParamsArray() As String)**

- Zkopíruje parametry z pole MachinesParamsArray do pole OutParamsArray

JobsInProgressClass

Proměnné

**Public Job As JobClass**

**Private MachinesStatusArray(10) As String**

- pole sloužící k uchování stavů rozpracovanosti zakázky na jednotlivých strojích

Metody

**Public Sub Initialize()**

**Public Sub Deinitialize()**

**Public Sub SetMachineJobStatus(MachineNr As Integer, status As String)**

- Uloží stav rozpracovanosit zakázky specifikovaného čísla stroje. Existují tři možné stavy, ve kterých se ta část zakázky (procesy), kterou tento stroj má zpracovat, může na-

cházet:

- waiting (čekající),
- active (stroj právě zakázku zpracovává) a
- finished (stroj dokončil všechny jemu určené dílčí procesy zakázky).

**Public Function GetMachineJobStatus(MachineNr As Integer) As String**

- Vrátí stav části zakázky specifikovaného čísla stroje

MachineClass

Proměnné

**Public Name As String**

**Public MachineStatus As String**

**Public SendingJobState As String**

**Public ControlJobState As String**

**Public JobActive As JobClass**

**Public JobNext As JobClass**

**Public ActiveJobNr As Integer**

**Private JDFProcessNameArray(CONFIG_PARAMS_MAX - 2) As String**

Metody

**Public Sub SetJDFProcessName(NrOfProcess As Integer, NameOfProcess As String)**

**Public Function GetJDFProcessName(NrOfProcess As Integer) As String**

**Public Function WriteParamsToOPC() As Boolean**

Poznámka:

Pro provázání mezi příslušnými stavovými diagramy stroje v programovatelném automatu a v řídicím systému je vyhrazena 16-ti bitová proměnná *General_ControlStation_Commands*. Jak v řídicím systému, tak v programovatelném automatu je definován význam jednotlivých bitů. Jejich význam je následující:

```
Private Const BIT_JobEnabled = 0          'r
Private Const BIT_ReadyForNewData = 1     'r
Private Const BIT_SendingNewData = 2      'r
Private Const BIT_NewDataSent = 3         'r
Private Const BIT_DataOK = 4              'r
Private Const BIT_MachineSettingOK = 5    'r
Private Const BIT_MachineReady = 6        'r
Private Const BIT_DataIncorrect = 7       'r
Private Const BIT_FreigabeEintaktung = 8  'w
```

- output z hlediska řídicího systému

```
Private Const BIT_NewJob = 9              'w
```

- output z hlediska řídicího systému

```
Private Const BIT_JobReadyToStartNew = 10 'r
Private Const BIT_JobInProgress = 11      'r
Private Const BIT_JobFinished = 12        'r
Private Const BIT_JobError = 13           'r
```

Tyto jednotlivé bity nejsou testovány či přiřazovány přímo, ale přes procedury (uvedené na příkladu bitu BIT_NewJob) nebo testovací funkce (uvedené na příkladu bitu BIT_DataOK) následující struktury

**Public Sub BitSetNewJob(SetToValue As Boolean)**

- Nastaví bit v proměnné *General_ControlStation_Commands* na místě definovaném hodnotou konstanty BIT_NewJob na hodnotu uvedenou v SetToValue.

**Public Function BitTestDataOK() As Boolean**

- Vrátí hodnotu bitu v proměnné *General_ControlStation_Commands* na místě definovaném hodnotou konstanty BIT_DataOK.

Ostatní funkce a proměnné jsou podobné struktury.

MyDBAccessClass

Privátní proměnné zde nejsou uvedeny z důvodů omezeného rozsahu této práce.

Metody

**Public Sub MoveFileFromTableToTable(FileToMove$, FromTable As String, ToTable As String)**

- Přesune zakázku (záznam v tabulce a také příslušný soubor v adresáři, který přísluší dané tabulce) z tabulky FromTable do tabulky ToTable.

**Public Function GetFileToSendFromTable() As String**

- Funkce vrátí název zakázky, která jako první v pořadí má být vyrobena z tabulky jobs_to_do. (Seřadí tedy nejdříve všechny záznamy v této tabulce vzestupně podle pole end_time, a vrátí první záznam)

**Public Sub ReadTransformedJdfTxtFile(FileToRead As String, ParamsArray() As String)**

- V poli uvedeném na místě parametru ParamsArray vrátí jednotlivé parametry zakázky načtené z JDF souboru. Ten není načten přímo, ale nejdříve je konvertován pomocí funkce ConvertJDFToMyTxt do souboru formátu CSV.
- Parametr FileToRead musí obsahovat pouze název souboru, cesta je doplněna na základě v třídě předem definovaných cest k adresářům.

**Public Function ConvertJDFToMyTxt(PathToFileToConvert As String)**

- Konvertuje JDF soubor uvedený jako parametr PathToFileToConvert do CSV souboru s názvem původního JDF souboru doplněném o příponu .txt (Tedy například z souboru job1.xml vytvoří CSV soubor job1.xml.txt)
- Parametr PathToFileToConvert musí obsahovat pouze název souboru, cesta je doplněna na základě v třídě předem definovaných cest k adresářům.

**Public Sub RunFileAsShell(FileToRun As String)**

- Spustí soubor uvedený v parametru FileToRun pomocí VBA funkce Shell.
- Parametr FileToRun musí obsahovat pouze název souboru, cesta je doplněna na základě předem v třídě definovaných cest k adresářům.

#### *4.2.4.4 Displeje v GraphWorX*

Stěžejní část mnou vytvořeného řešení je založena na pěti displejích.

První - *Control Panel* – je vlastně přepínací panel, který především přepíná (zobrazuje) další displeje.

Druhý – J*obs Prepared* - představuje operátorské rozhraní pro správu zakázek a všeho co s nimi souvisí.

Třetí - *Jobs To Do* - pak umožňuje určování pořadí výroby zakázek operátorem.

Čtvrtý – P*roduction Control* – umožňuje sledování průběhu výroby zakázek a rovněž na jeho pozadí běží řízení procesu výroby.

Pátý – *Jobs Finished* – slouží k zobrazení vyrobených zakázek.

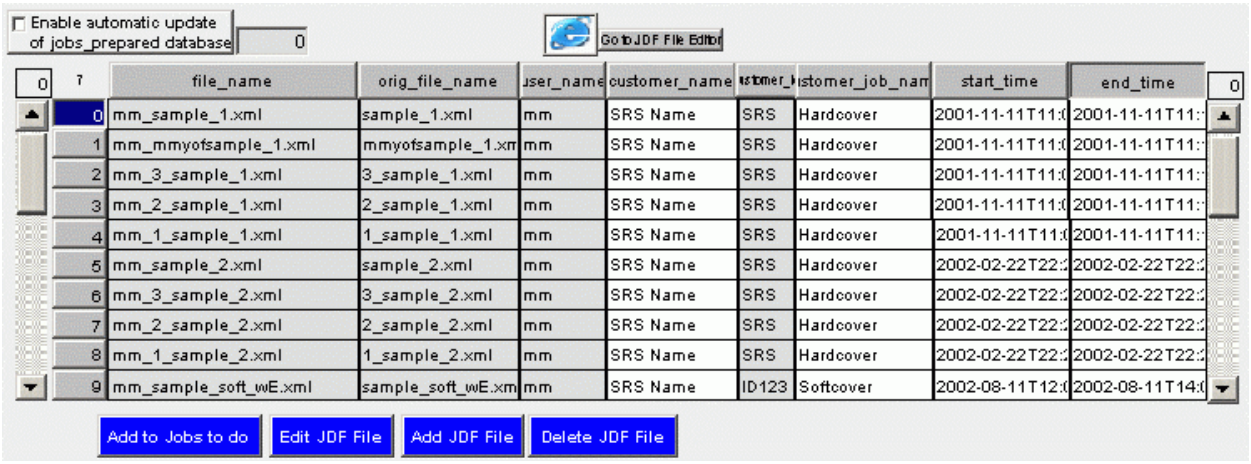
4.2.4.4.1 Přepínací panel

Tento displej je zobrazen po zapnutí řídicího systému. Automaticky rovněž při přechodu do runtime režimu spustí displej production control (nastartuje tím tak řízení a monitorování výroby).


*Obr. 4.2-11 Výřez displeje „přepínací panel“*

4.2.4.4.2 Jobs prepared

Ústřední částí tohoto displeje je tabulka znázorňující připravené zakázky – jobs prepared – která je pomocí technologie Microsoft ADO a přes ODBC rozhraní provázána s tabulkou *jobs_prepared* v Microsoft Access databázi *db_jobs.mdb*.




*Obr. 4.2-12 Část displeje „Jobs Prepared“ v GraphpWorX*

Tabulka databáze Access není zobrazena přímo, ale jsou načteny jednotlivé hodnoty polí (sloupců) v řádce. Protože by bylo nepřehledné uvádět programový kód v jazyce VBA přímo na tomto místě, je možné nalézt kód, který načte data z tabulky databáze v části 7.3.1.1. Ve stručnosti lze popsat jeho funkci následovně:

- nejdříve se vytvoří objekt typu "ADODB.Connection", díky jemuž je možné přistupovat pomocí ODBC rozhraní k existujícímu DNS zdroji (definovaném na daném počítači) a objekt typu "ADODB.Recordset", který pak představuje výsledek SQL dotazu vyvolaného příkazem open. Tedy část kódu:

```
Set conn = CreateObject("ADODB.Connection")
Set rs = CreateObject("ADODB.Recordset")
conn.Open DSN, UserName, Password
rs.Open "SELECT * FROM [" & DBTable & "] ORDER BY [" & OrderByFieldName &
"] ASC", conn, 1, 3
```

- objekt rs pak představuje výsedek sql dotazu (v daném případě celou tabulku). Protože

není možné zobrazit všechny záznamy tabulky na displej najednou, je vybráno vždy jen deset položek z tabulky a to od určitého záznamu. Pozice počátečního zobrazovaného záznamu (offsetu) je určována hodnotou proměnné start_offset, kterou lze libovolně nastavit v mezích závislých na celkovém počtu řádků v tabulce a počtem zobrazovaných řádků.

Podobný kód pro načtení záznamů z tabulky využívají (volají) i další definované procedury, díky nimž je pak možno provádět následující operace s tabulkou.

- zobrazované **záznamy** je možno **třídit** dle libovolného z zobrazených polí, a to vzestupně. Po kliknutí na příslušné tlačítko v hlavičce tabulky s názvem pole pak dojde k překreslení zobrazené tabulky setříděné podle daného pole. To je realizováno přiřazením příslušného názvu pole za proměnnou OrderByFieldName do řetězce SQL dotazu uvedeného výše. Standardně je pak vždy zvoleno řazení podle doby, kdy má být zakázka hotova.
- u některých polí je pak možno ještě **měnit hodnoty** jednotlivých **záznamů**. Tímto způsobem je pak možno například měnit pořadí zakázek změnou hodnoty času, kdy má být zakázka hotova.

S jednotlivými záznamy, které představují vlastní JDF soubory, je pak pomocí tlačítek umístěných pod tabulkou dále **možné provádět** ještě **následující operace**:

- **přesunout zakázku** z tabulky jobs prepared do tabulky jobs to do, a tedy de facto dát **pokyn** k samotné **výrobě zakázky**, neb v tabulce jobs to do je zařazena do fronty zakázek určených k zpracování.
- **editovat** hodnoty parametrů jednotlivých specifikací výroby knihy (zakázky), což znamená **měnit hodnoty atributů** jednotlivých elementů **JDF** souborů. To je realizováno pomocí zavolání formuláře Visual Basic, která obsahuje ActiveX objekt Microsoft Web Browser. V něm je zobrazena html stránka dynamicky vygenerovaná s pomocí webového serveru Apache a skriptu jazyka PHP. Je využit stejný skript jaký využívá dříve popsaná webová aplikace pro práci s JDF soubory, konrétně JDF File Ediotr, je ovšem volán poněkud odlišným způsobem a tím tedy obsahuje poněkud omezené možnosti – načtený soubor parametrů například lze uložit pouze pod stejným názvem a nelze se dostat do JDF File Exploreru.
- **přidat JDF** soubor pomocí standardního dialogového okna pro otevření souboru.
- **vymazat** záznam z tabulky

Kromě možnosti přidání JDF souboru manuálně je dále možné aktivovat takzvaný automatický režim aktualizace tabulky *jobs prepared* (tlačítko Enable automatic update of jobs_preperad database). V automatickém režimu je s předem definovanou periodou (10 s) spuštěn VBA skript, který pomocí vytvořené funkce `RunFileAsShell` spustí dávkový soubor `u_gen_update_db.php.bat` . Příslušné volání této funkce je:

```
Call RunFileAsShell("u_gen_update_db.php.bat")
```

Dávkový soubor obsahuje následující příkaz pro zavolání PHP skriptu:

```
@c:\apache\php\php-cli.exe -q -c c:\apache\php\cli-ini
E:\MOJE\apache_root\JDF\gen_update_db.php
```

Zavolá tedy skript *gen_update_db.php* způsobem volání PHP z příkazové řádky. Tento skript využívá možností ODBC rozhraní a aktualizuje tabulku *jobs_prepared* v MS Access databázi.

Přistupuje k předem definovanému adresáři *job_list*, který obsahuje podadresáře stejné struktury jako domovské adresáře jednotlivých uživatelů webové aplikace pro práci s JDF soubory. Tyto podadresáře slouží k ukládání souborů (zakázek), které byly poslány jednotlivými uživateli z webové aplikace k vyhotovení. Jedná se tedy o jakési rozhraní mezi webovou aplikací a řídicím systémem výrobní linky. Navíc je zde jeden podadresář *gen*, který je určen pro přidávání JDF souborů přímo z GraphWorX.

Skript gen_update_db.php nejdříve postupně projde všechny zmíněné podadresáře. Pro každý podadresář načte do dvojrozměrného pole důležité informace o obsažených souborech. První dimenze je rovna počtu souborů a pro každý soubor jsou pak do druhé dimenze uloženy následujícící informace:

- jméno souboru
- jméno zákazníka
- identifikační číslo zákazníka
- název zakázky
- počáteční čas výroby
- doba dokončení zakázky

Příklad části tohoto PHP skriptu:

```
$A_file_tab[$files_count]["name"]              = $sub_entry;
$A_file_tab[$files_count]["customer_name"]     = $JobD_CustomerName;
$A_file_tab[$files_count]["customer_id"]       = $JobD_CustomerID;
$A_file_tab[$files_count]["customer_job_name"] = $JobD_CustomerJobName;
$A_file_tab[$files_count]["start_time"]        = $JobD_Start;
$A_file_tab[$files_count]["end_time"]          = $JobD_End;
```

Poté pro všechny soubory v podadresáři (prvky pole) zkontroluje, zda soubor s příslušným názvem je obsažen v tabulce *jobs_prepared* a v záporném případě (soubor tedy ještě není přítomen) vloží nový záznam s výše zmíněnými informacemi, které odpovídají struktuře jednotlivých polí.

#### 4.2.4.4.3 Jobs to do

Ústřední části tohoto displeje je opět tabulka s jednotlivými zakázkami. Tentokrát jsou to zakázky, které čekají na vyrobení, napojená podobně jako je popsáno v předešlé části na databázi MS Access, tentokrát ale svázána s tabulkou *jobs_to_do*.

Podobně jako s tabulkou jobs*_prepared* je i zde možno provádět určité operace s jednotlivými záznamy, které představují vlastní JDF soubory, možnosti jsou ale poněkud odlišné:

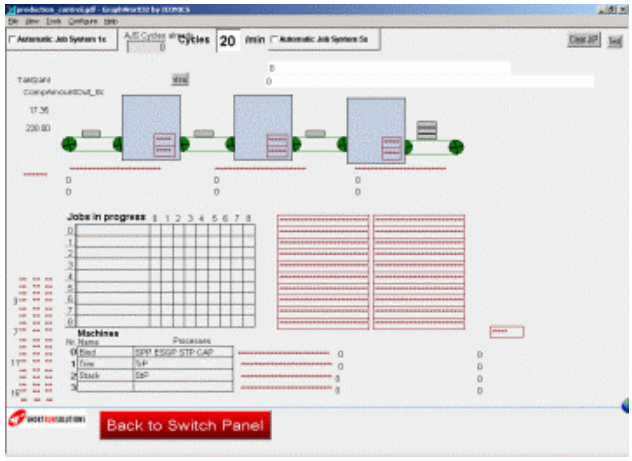
- měnit parametry zakázky, ale již jen pouze ty důležité, uvedené přímo v tabulce (měnit hodnoty jednotlivých záznamů). Tímto způsobem je možno například měnit pořadí zakázek změnou hodnoty času, kdy má být zakázka hotova. Editovat hodnoty parametrů JDF souborů již nění možné.
- přesunout zakázku zpět do tabulky *jobs_prepared,* a tedy tím ji vymazat z tabulky *jobs_to_do.*

#### 4.2.4.4.4 Production Control (jobs_in_progress)

Tento displej představuje rozhraní mezi operátorem výrobní linky a programem řídicího počítače. Vlastní řízení celého výrobního procesu se odehrává na pozadí tohoto displeje. Zakázky, které jsou ve výrobě, jsou jednak uloženy v tabulce jobs_in_progress, ale také jsou všechny jejich informace uloženy v poli objektů OJobs (viz 4.2.4.3.2), které využívá pak program pro řízení strojů.

V tomto displeji je zobrazena vizualizace celé linky. Operátor může provádět operace ovlivňující proces výroby knihy

Vlastní řízení procesu, které již není přímo ovlivňováno operátorem, využívá vlastností a metod objektů popsaných již v části 4.2.4.3. Postup řízení je pak podrobně popsán v následující části 4.2.4.5.




*Obr. 4.2-13 Displej „Production Control“*

4.2.4.4.5 Jobs Finished

Displej je založen na tabulce jobs_finished. V této tabulce jsou zobrazeny vyrobené zakázky. Dále je možné archivovat zakázky aby zbytečně nezabíraly místo v této tabulce.

### *4.2.4.5 Řízení strojů*

Vlastní jádro řídicího systému probíhá na pozadí displeje Jobs_to_do. Program řídicího systému linky je složen ze skriptů v jazyce VBA, využívajících výše uvedené definované třídy a jejich vlastnosti a metody.

4.2.4.5.1 Vytvoření konfigurace linky

Výhody, které přináší možnost definovat dynamicky aktuální konfiguraci linky bez nutnosti zasahovat do vlastního programového kódu v VBA, jsou vcelku zřejmé. Možnost jednoduše změnit konfiguraci linky (změnou externího konfiguračního souboru) například v případě rozšíření linky či poruchy stroje je z určitého pohledu dokonce nutná.

Proto jsem definoval strukturu konfiguračního souboru, který v této první fázi je založen na souboru formátu CSV. Zatím chybí kontrola smysluplnosti zadané konfigurace z hlediska technologického procesu výroby knihy.

Variantě základní konfigurace linky sestávající se z následujících čtyř strojů:

- lepicí automat - s procesy opracování hřbetu, lepení předsádky, lepení knihařského gázu a aplikace obalu
- sušička
- řezačka – s procesem ořezání
- balící stroj – s procesem naskládání knih do balíků,

odpovídá následující konfigurační soubor `machines_config.cfg`

```
# ****** MACHINES CONFIG ****************************************
# This file is used for configuration of the production line machines
# The character # at the beginning of the line is for comment
# Config sturcture:
# MachineNr;Machine_Type;Process1[;Process2;Process3;Process4]
0;Bind;SpinePreparation;EndSheetGluing;SpineTaping;CoverApplication
1;Dry;
2;Trim;Trimming
3;Stack;Stacking
```

Tento soubor je při přechodu do režimu runtime načten do pole a na jeho základě vytvořeny jednotlivé instance pole Machines typu MachineClass s příslušnými parametry odpovídajícími konfiguračnímu souboru (procedura viz 7.3.1.1 na straně 110).

Do budoucna je jistě zajímavá myšlenka definice konfiguračního souboru ve formátu XML, pro který by bylo možné definovat patřičné schéma. Kromě větší přehlednosti souboru by ve schématu byly definovány různé možnosti platné konfigurace linky, a tím tedy i usnadněna kontrola jeho platnosti. Jen pro úplnost uvádím možný návrh takového XML konfiguračniho souboru odpovídajícího výše uvedenému CSV[1] souboru:

```
<?xml version="1.0" encoding="UTF-8"?>
<SREProductionLineConfiguration>
      <Machines>
            <MachineBind Nr=”0”>
                  <ProcessName>SpinePreparation</ProcessName>
                  <ProcessName>EndSheetGluing</ProcessName>
                  <ProcessName>SpineTaping</ProcessName>
                  <ProcessName>CoverApplication</ProcessName>
            </MachineBind>
            <MachineDry Nr=”1”>
            </MachineDry>
            <MachineTrim Nr=”2”>
                  <MachineName>Trim</MachineName>
                  <ProcessName>Trimming</ProcessName>
            </MachineTrim>
            <MachineStack Nr=”3”>
                  <ProcessName>Stacking</ProcessName>
            </MachineStack>
      </Machines>
</SREProductionLineConfiguration>
```

#### 4.2.4.5.2 Cyklické řízení

Vlastní řízení procesu zajišťuje již zmíněná metoda `CyclicProductionControl()` objektu ControlSystem, inicializovaného po přepnutí do runtime režimu jako instance třídy ControlSystemClass.

Metoda `CyclicProductionControl` je volána periodicky[2] každých 1200 ms. Jak již bylo stručně uvedeno v části 4.2.4.3, provádí několik hlavních činností:

- zařazuje novou zakázku do výroby
- řídí všechny stroje (metoda ControlMachine)
- v případě, že je zakázka vyrobena (hotova)., přesouvá ji do tabulky dokončených zakázek
- zobrazuje aktuální stav zakázek

[1] CSV soubor – soubor dat strukturovaných dle určitého pravidla oddělených předem definovaným oddělovacím znakem, většinou čárkou, často však například středníkem

[2] Určení této periody vychází z maximální předpokládané frekvence taktu strojů v navrhované lince, která je 20 taktů / min. To znamená, že v maximálním případě může být každé 3 vteřiny vyrobena jedna kniha. Doba 1,2 s pak splňuje s rezervou známou podmínku Shanonova teorému.

Před rozborem jednotlivých činností je ale nejdříve nutné definovat stavy, ve kterých se zakázka, respektive její část z hlediska technologického postupu výroby, může na jednotlivých strojích nacházet. Každý stroj v lince totiž vykonává předem daný soubor technologických procesů. Po úspěšném provedení všech procesů na všech strojích je pak zakázka dokončena.

Možné stavy části zakázky na jednotlivých strojích byly již obecně definovány v části 4.2.4.3.2, přesto však ještě jednou podrobněji:

- waiting (čekající) – zakázka je již zařazena v tabulce jobs_in_progress, ještě ale není daný stroj na ní připraven (například zpracovává předchozí zakázku)
- active (stroj právě zakázku zpracovává) - stroj dostal povel od řídicího systému připravit se na novou zakázku, nastavuje tedy své mechanické části dle nových dat, nebo již probíhá výroba zakázky
- finished (stroj dokončil dané procesy zakázky) – stroj dostal od řídicího systému povel ukončit zakázku, neboť počet vyrobených produktů, které hlásí na řídicí systém, dosáhl požadovaného počtu

Kromě těchto stavů je pak ještě nutné definovat podmínky, za kterých dochází k jejich změně a to především v souvislosti se stavem předchozích či následujících strojů.

Novou zakázku je možné zařadit do právě vyráběných zakázek tehdy, pokud první stroj v lince (s indexem 0 v poli Machines) je připraven na novou zakázku. Tato situace nastane pokud:

- v poli právě vyráběných zakázek není žádná zakázka, nebo
- po první nalezené zakázce ve stavu active u stroje s indexem 0 není zařazena žádná další zakázka (tedy žádná není ani ve stavu waiting), nebo
- v poli právě vyráběných zakázek je u stroje s indexem 0 (poslední) zakázka ve stavu finished a za ní není přítomna žádná další zakázka,

Pokud je splněna některá z těchto podmínek, je možné zařadit zakázku do jobs_in_progess. Současně dojde k nastavení stavů nové zakázky u všech strojů na waiting.

Přechod do stavu active

Zakázka N-tého stroje může přejít (ze stavu waiting) do stavu active pouze tehdy, pokud:

- není žádná zakázka ve stavu active a existuje zakázka ve stavu waiting, nebo
- byla právě dokončena předchozí zakázka a následující je ve stavu waiting

Přechod do stavu finished

Zakázka N-tého stroje může přejít do stavu finished pouze tehdy, pokud:

- počet vyrobených produktů (hlášených strojem řídicímu systému) dosáhne požadované hodnoty uvedené v parametrech zakázky – řídicí systém pak pošle signál k zablokování vstupního kanálu stroje a provede potřebné operace, nebo
- v případě, že z různých důvodů není možné vyrobit požadovaný počet produktů (například došlo-li ke kolizi v některém z předchozích strojů a k výrobě zmetku, který byl odstraněn), dosáhne-li počet vstupních produktů do stroje aktuálního maximálního možného počtu dostupných produktů – v tomto případě ale není zakázka 100% splněna a musí dojít k zaznamenání této situace řídicím systémem a v ideálním případě později k automatické nápravě.

V případě, že zakázka se u všech strojů nachází ve stavu finished, je automaticky přesunuta do tabulky jobs_finished.

Modelový příklad průběhu výroby 5 zakázek při konfiguraci na 4 strojů

Díky tomu, že každý prvek pole OJobs má definované vlastní pole MachinesStatusArray, vzniká jakési dvojrozměrné pole, umožňující přiřazování stavu rozpracovanosti jednotlivých zakázek právě vyráběných na všech strojích. Při počtu N strojů v lince může být teoreticky aktiv-

ních N zakázek (a v tabulce jobs_in_progres pak N+1 zakázek – zakázka +1 pouze ve stavu waiting). Uveďme na příkladu čtyř strojů časový průběh výroby pěti zakázek.

**Fáze 1** (žádná zakázka není právě vyráběna, 5 zakázek čeká v tabulce jobs_to_do)

| | | *Stav rozpracovanosti zakázky na stroji* | | | |
|---|---|---|---|---|---|
| *OJob* | *Jméno zakázky* | *Stroj 0* | *Stroj 1* | *Stroj 2* | *Stroj 3* |
| *0* | | | | | |
| *1* | | | | | |
| *2* | | | | | |
| *3* | | | | | |
| *4* | | | | | |

**Fáze 2** (první zakázka přesunuta to tabulky jobs_in_progress, příprava zakázky na výrobu, 4 zakázky zbývají v tabulce jobs_to_do)

| | | *Stav rozpracovanosti zakázky na stroji* | | | |
|---|---|---|---|---|---|
| *OJob* | *Jméno zakázky* | *Stroj 0* | *Stroj 1* | *Stroj 2* | *Stroj 3* |
| *0* | Zakázka_1 | W | W | W | W |
| *1* | | | | | |
| *2* | | | | | |
| *3* | | | | | |
| *4* | | | | | |

**Fáze 3** (zahájena výroba první zakázky na stroji 0, zakázka 2 čeká na výrobu, 3 zakázky zbývají v tabulce jobs_to_do)

| | | *Stav rozpracovanosti zakázky na stroji* | | | |
|---|---|---|---|---|---|
| *OJob* | *Jméno zakázky* | *Stroj 0* | *Stroj 1* | *Stroj 2* | *Stroj 3* |
| *0* | Zakázka_1 | A | W | W | W |
| *1* | Zakázka_2 | W | W | W | W |
| *2* | | | | | |
| *3* | | | | | |
| *4* | | | | | |

**Fáze 4** (zahájena výroba první zakázky na stroji 1, zakázka 2 čeká na výrobu, 3 zakázky zbývají v tabulce jobs_to_do)

| | | *Stav rozpracovanosti zakázky na stroji* | | | |
|---|---|---|---|---|---|
| *OJob* | *Jméno zakázky* | *Stroj 0* | *Stroj 1* | *Stroj 2* | *Stroj 3* |
| *0* | Zakázka_1 | A | A | W | W |
| *1* | Zakázka_2 | W | W | W | W |
| *2* | | | | | |
| *3* | | | | | |
| *4* | | | | | |

**Fáze 5** (dokončena výroba první zakázky na stroji 0 a1, započata zakázka 2 na stroji 0, zakázka 3 čeká, 2 zakázky zbývají v tabulce jobs_to_do)

| | | *Stav rozpracovanosti zakázky na stroji* | | | |
|---|---|---|---|---|---|
| *OJob* | *Jméno zakázky* | *Stroj 0* | *Stroj 1* | *Stroj 2* | *Stroj 3* |
| *0* | Zakázka_1 | F | F | A | A |
| *1* | Zakázka_2 | A | A | W | W |
| *2* | Zakázka_3 | W | W | W | W |
| *3* | | | | | |
| *4* | | | | | |

**Fáze 6** (dokončena výroba první zakázky na stroji 2, započata zakázka 2 na stroji 2, 1 zakázka zbývá v tabulce jobs_to_do)

| | | *Stav rozpracovanosti zakázky na stroji* | | | |
|---|---|---|---|---|---|
| *OJob* | *Jméno zakázky* | *Stroj 0* | *Stroj 1* | *Stroj 2* | *Stroj 3* |
| *0* | Zakázka_1 | F | F | F | A |
| *1* | Zakázka_2 | F | F | A | W |
| *2* | Zakázka_3 | A | A | W | W |
| *3* | Zakázka_4 | W | W | W | W |
| *4* | | | | | |

**Fáze 7** (dokončena výroba první zakázky na stroji 3, započata zakázka 4 na stroji 0, 0 zakázek v tabulce jobs_to_do)

| | | *Stav rozpracovanosti zakázky na stroji* | | | |
|---|---|---|---|---|---|
| *OJob* | *Jméno zakázky* | *Stroj 0* | *Stroj 1* | *Stroj 2* | *Stroj 3* |
| *0* | Zakázka_1 | F | F | F | F |
| *1* | Zakázka_2 | F | F | F | A |
| *2* | Zakázka_3 | F | A | A | W |
| *3* | Zakázka_4 | A | W | W | W |
| *4* | Zakázka_5 | W | W | W | W |

**Fáze 8** (zakázka 1 byla přesunuta do tabulky jobs_finished, žádná zakázka v tabulce jobs_to_do)

| | | *Stav rozpracovanosti zakázky na stroji* | | | |
|---|---|---|---|---|---|
| *OJob* | *Jméno zakázky* | *Stroj 0* | *Stroj 1* | *Stroj 2* | *Stroj 3* |
| *0* | Zakázka_2 | F | F | F | A |
| *1* | Zakázka_3 | F | F | A | W |
| *2* | Zakázka_4 | A | A | W | W |
| *3* | Zakázka_5 | W | W | W | W |
| *4* | | | | | |

**Fáze 9** (žádná zakázka v tabulce jobs_to_do)

| | | *Stav rozpracovanosti zakázky na stroji* | | | |
|---|---|---|---|---|---|
| *OJob* | *Jméno zakázky* | *Stroj 0* | *Stroj 1* | *Stroj 2* | *Stroj 3* |
| *0* | Zakázka_2 | F | F | F | F |
| *1* | Zakázka_3 | F | F | F | A |
| *2* | Zakázka_4 | F | F | A | W |
| *3* | Zakázka_5 | A | A | W | W |
| *4* | | | | | |

**Fáze 10** (žádná zakázka v tabulce jobs_to_do)

| | | *Stav rozpracovanosti zakázky na stroji* | | | |
|---|---|---|---|---|---|
| *OJob* | *Jméno zakázky* | *Stroj 0* | *Stroj 1* | *Stroj 2* | *Stroj 3* |
| *0* | Zakázka_3 | F | F | F | A |
| *1* | Zakázka_4 | F | F | A | W |
| *2* | Zakázka_5 | F | A | W | W |
| *3* | | | | | |
| *4* | | | | | |

**Fáze 11** (žádná zakázka v tabulce jobs_to_do)

| | | *Stav rozpracovanosti zakázky na stroji* | | | |
|---|---|---|---|---|---|
| *OJob* | *Jméno zakázky* | *Stroj 0* | *Stroj 1* | *Stroj 2* | *Stroj 3* |
| *0* | Zakázka_3 | F | F | F | F |
| *1* | Zakázka_4 | F | F | F | A |
| *2* | Zakázka_5 | F | A | A | W |
| *3* | | | | | |
| *4* | | | | | |

**Fáze 12** (žádná zakázka v tabulce jobs_to_do)

| | | *Stav rozpracovanosti zakázky na stroji* | | | |
|---|---|---|---|---|---|
| *OJob* | *Jméno zakázky* | *Stroj 0* | *Stroj 1* | *Stroj 2* | *Stroj 3* |
| *0* | Zakázka_4 | F | F | F | A |
| *1* | Zakázka_5 | F | F | A | W |
| *2* | | | | | |
| *3* | | | | | |
| *4* | | | | | |

**Fáze 13** (žádná zakázka v tabulce jobs_to_do)

| | | *Stav rozpracovanosti zakázky na stroji* | | | |
|---|---|---|---|---|---|
| *OJob* | *Jméno zakázky* | *Stroj 0* | *Stroj 1* | *Stroj 2* | *Stroj 3* |
| *0* | Zakázka_5 | F | F | F | A |
| *1* | | | | | |
| *2* | | | | | |
| *3* | | | | | |
| *4* | | | | | |

**Fáze 14** (žádná zakázka v tabulce jobs_to_do)

| | | *Stav rozpracovanosti zakázky na stroji* | | | |
|---|---|---|---|---|---|
| *OJob* | *Jméno zakázky* | *Stroj 0* | *Stroj 1* | *Stroj 2* | *Stroj 3* |
| *0* | Zakázka_5 | F | F | F | F |
| *1* | | | | | |
| *2* | | | | | |
| *3* | | | | | |
| *4* | | | | | |

**Fáze 15** (všechny zakázky hotové, žádná zakázka v tabulce jobs_to_do)

| | | *Stav rozpracovanosti zakázky na stroji* | | | |
|---|---|---|---|---|---|
| *OJob* | *Jméno zakázky* | *Stroj 0* | *Stroj 1* | *Stroj 2* | *Stroj 3* |
| *0* | | | | | |
| *1* | | | | | |
| *2* | | | | | |
| *3* | | | | | |
| *4* | | | | | |

Start nové zakázky (zařazení do jobs_in_progress)

Novou zakázku je možné zařadit do právě vyráběných zakázek v případě splnění výše zmíněných podmínek. V tom případě dojde k vybrání té zakázky z tabulky jobs_to_do, která má být vyrobena nejdříve, a k jejímu zařazení do jobs_in_progress.

Řízení strojů

Řízení průběhu výroby zakázky na jednotlivých strojích je zřejmě klasickým sekvenčním řízením. Pro jeho realizaci je nutné jednotlivé činnosti provádět dle sekvenčního funkčního diagramu. Vzhledem k tomu, že na řízení procesu se v podstatě podílejí dva programy – jeden v rámci nadřazeného řídicího systému a druhý v rámci programovatelného automatu – je nutné definovat dva sekvenční diagramy, jejichž stavy budou navzájem provázány.

Provázání jsem realizoval přes 16-ti bitovou OPC proměnnou

`(název_stroje)_General_ControlStation_Commands`

Přiřazení jednotlivých bitů pro řídicí systém je uvedeno v části 4.2.4.3. Stejné přiřazení pak rovněž musí být v programu automatu. Z hlediska řídicího systému jsou jako výstupy následující bity:

- BIT_FreigabeEintaktung – jeho nastavením na 0 dojde k virtuálnímu zablokování vstupního kanálu pro produkty do stroje; pokud je roven 1, může stroj po přijetí produktu po vstupním pásu vyrábět.
- BIT_NewJob – jeho nastavení na 1 je signál pro stroj, že současná zakázka musí být ukončena a má se připravit na další

Ostatní bity jsou pro řídicí systém vstupy. Pro programovatelný automat je to pochopitelně obráceně. Význam zbývajících důležitých bitů:

- BIT_DataOK – hlášení PA pro řídicí systém, že data nové zakázky jsou platná[1] (jsou v souladu s konstrukčními a technologickými omezeními)
- BIT_DataIncorrect - hlášení PA pro řídicí systém, že data nové zakázky nejsou platná
- BIT_MachineReady - hlášení PA pro řídicí systém, že stroj je nastaven a připraven na novou zakázku

Na obr. 4.2-14 je uvedena legenda k sekvenčním funkčním diagramům, jejichž standardní značení jsem pro snazší zápis odpovídající přímo proměnným v programu v jazyce VBA a B&R Automation Basic poněkud modifikoval.

[1] Data posílaná jednotlivým strojům jsou již samozřejmě otestována dle aktuální konfigurace linky a z toho plynoucích technologických omezení. Řídící systém před nastavením bitu NewJob na 1 (pokyn k přípravě na novou zakázku) rovněž kontroluje, zda data poslaná na PA jsou již skutečně uložena v daných proměnných. Kontrola na straně PA je teoreticky nadbytečná, avšak prakticky nutná.


*Obr. 4.2-14 Legenda k funkčním diagramům*

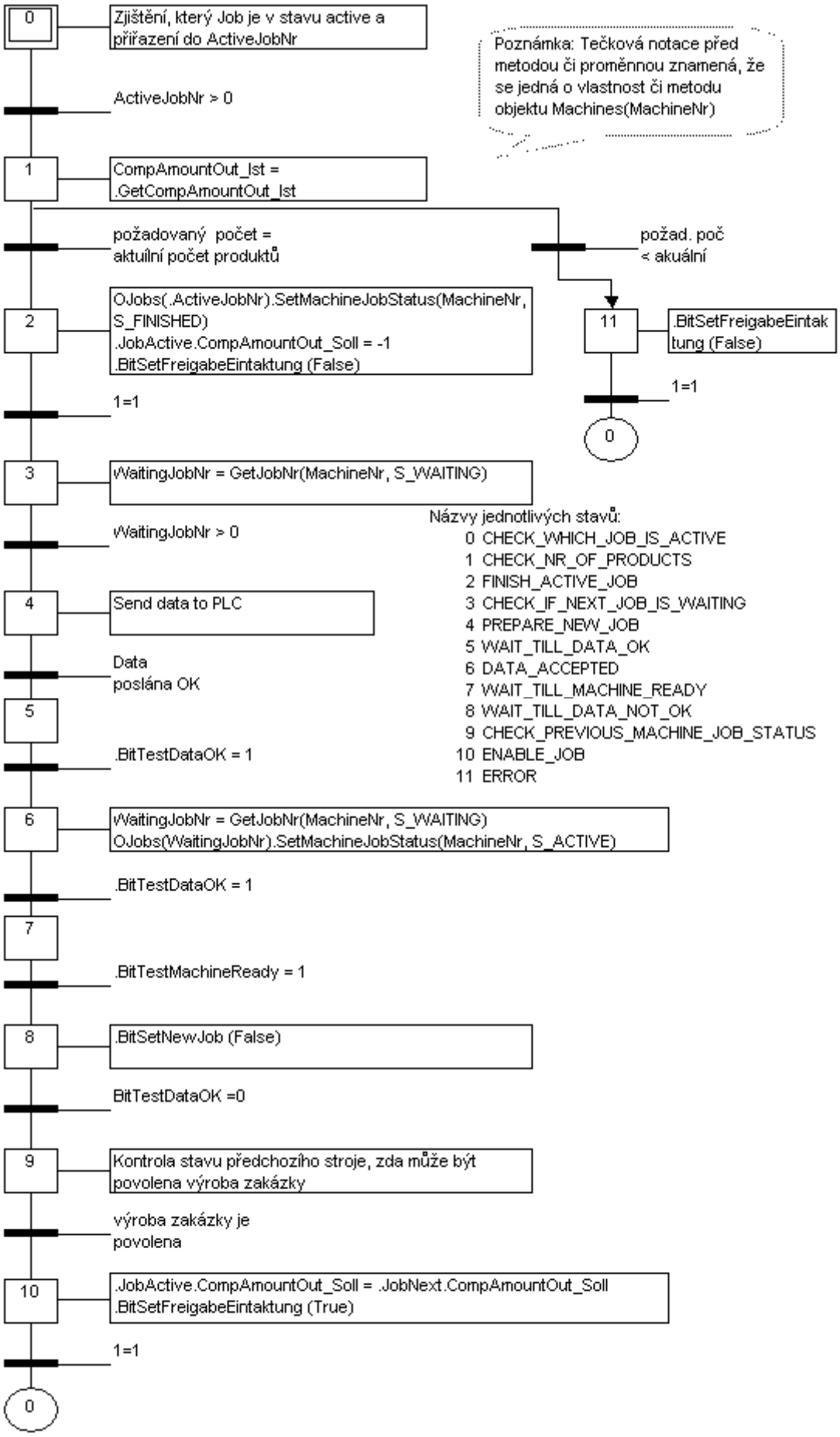
Diagram řídicího systému pro komunikaci programovatelným automatem

Diagram řídicího systému je uveden na obr. 4.2-15. Krokový diagram je takto definován v metodě ControlMachine objektu ControlMachine. Díky objektovému přístupu je pak tento funkční diagram použit pro každý stroj – pro všechny objekty pole Machines. Obecně je definován v třídě ControlSystemClass v metodě ControlMachine.

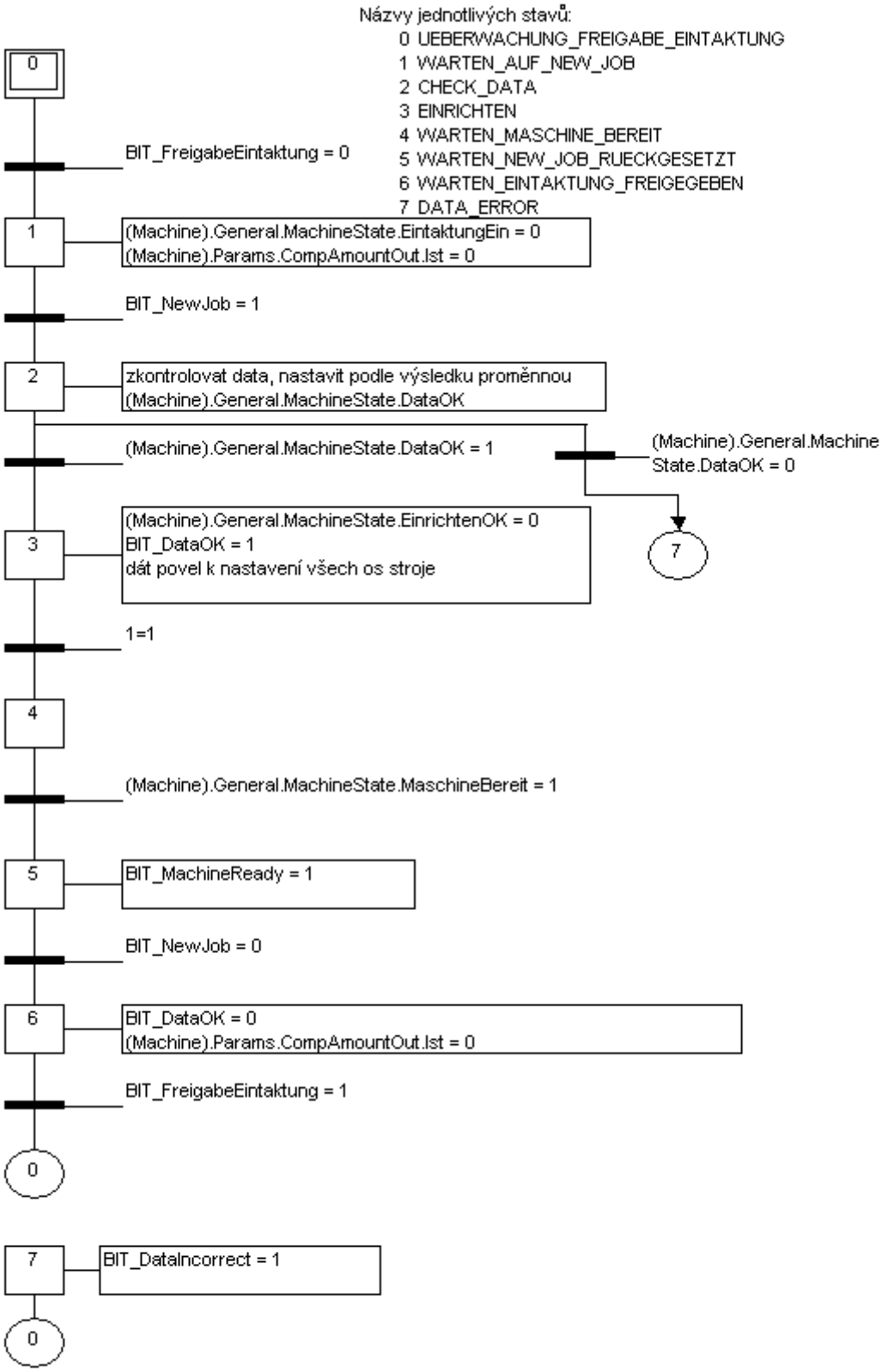
Tato metoda má jako parametry jednak pole strojů (instance třídy MachineClass) a rovněž číslo stroje, který má řídit. Pro řízení stavů je využita proměnná ControlJobState objektu Machines(MachineNr). Jak již bylo uvedeno v části 4.2.4.3, metoda ControlMachine využívá metody objektu Machines(MachineNr) k přístupu k OPC proměnným a tím tedy k vlastnímu řízení stroje.

Diagram programovatelného automatu pro komunikaci s řídicím systémem

Funkční diagram pro programovatelný automat je na obr. 4.2-16.




*Obr. 4.2-15 Funkční diagram – řídicí systém*




*Obr. 4.2-16 Funkční diagram – programovatelný automat*

## *4.3 Programovatelné automaty*

Návrh programové ralizace řízení programovatelných automatů řídicím systémem a návrh vlastního programu PA spolu nepochybně úzce souvisí. Funkční diagramy uvedené na obr. 4.2-15 a obr. 4.2-16 jsou spolu provázány a je tedy vhodné je navrhnout s ohledem prvního na druhý a naopak. Rovněž definice OPC proměnných by měla zohledňovat potřeby jak řídicího systému tak PA.

V případě, že je tímto způsobem definována komunikace a jednotlivé stavy řízení, není pak již nezbytně nutné, aby tvorba programových kódů (obecně v odlišných programových prostředích) pro programovatelné automaty a řídicí systémem byla nějak na sobě závislá.

Protože tu část programového kódu PA, určenou pro komunikaci s nadřazeným řídicím systémem dle funkčních diagramů, jsem vytvořil rovněž v rámci této práce, ukázalo se, že paralelní návrh kódu řídicího systému a PA přináší přesto určité výhody. Například při definici OPC proměnných (viz 4.2.4.2.4) jsem mohl s výhodou online importovat do konfigurátoru OPC serveru proměnné definované v PA. Dále pak ve fázi vývoje a průběžného testovaní realizace funkčního diagramu bylo možné přímo zkoušet, do jaké míry uspokojuje průběžné řešení funkční nároky a následně toto řešení vylepšovat.

V části 4.3.1 je tedy dále uveden popis struktury definovaných proměnných pro PA a v části 4.3.2 pak kód v jazyce B&R Automation Basic pro realizaci funkčního diagramu na příkladu stroje „Stack“ – balícího stroje.

### 4.3.1 Struktura definovaných proměnných pro přenos dat (přehled definovaných proměnných založených na JDF a určených pro komunikaci mezi programovatelnými automaty a řídicím systémem, respektive jeho OPC serverem)

Automation Studio nabízí možnost definice vlastních datových typů a proto pro definici parametrů výrobního procesu jsem samozřejmě využil této možnosti. Kromě usnadnění definice proměnných tato možnost značně zpřehledňuje a především zjednodušuje práci s parametry jednotlivých strojů.

V dalších částech jsou postupně uvedeny struktury tří vytvořených proměnných, které využívají právě vlastních definovaných typů – OPC_Bind typu OPC_BIND, OPC_Trim typu OPC_TRIM a OPC_Stack typu OPC_STACK.

Tuto znázorněnou přehlednou stromovou strukturu proměnných jsem získal s využitím volby „Watch“ Automation Studia (uvedeny jsou pouze první dva sloupce obsahující název proměnné a její typ, ostatní sloupce watch okna, jako například aktuální hodnota proměnné jsem pro přehlednost vynechal).

Pro úplnost je u některých procesů ještě uveden obrázek z JDF specifikace, vysvětlující význam parametrů jednotlivých procesů.

#### *4.3.1.1 OPC_Bind*

4.3.1.1.1 .General (obecné parametry)

| *Název* | *Typ* | |
|---|---|---|
| OPC_Bind | OPC_BIND | |
| General | MACHINE_GENERAL | |
| Job | JOB | |
| Name | STRING(30) | |
| Status | STRING(15) | |
| DataStatus | STRING(10) | |
| MachineState | MACHINE_STATE | |
| Language | STRING(3) | |
| Error | STRING(20) | |
| EinrichtenMoeglich | BOOL | |
| EinrichtenOK | BOOL | |
| StartJobMoeglich | BOOL | |
| DataOK | BOOL | |
| EintaktungEin | BOOL | |
| MaschineBereit | BOOL | |
| ControlStation | CONTROL_STATION | [1] |
| Commands | UINT | |
| Params | BINDINGPARAMS | |

4.3.1.1.2 Params

*4.3.1.1.2.1 Components (Rozměry a počty vstupních a výstupních produktů procesů)*

| | | |
|---|---|---|
| OPC_Bind | OPC_BIND | |
| General | MACHINE_GENERAL | |
| Params | BINDINGPARAMS | |
| CompDimIn | COMPDIM | vstupní produkt |
| Width | REAL | - šířka |
| Height | REAL | - výška |
| Thickness | REAL | - tloušťka |
| CompDimOut | COMPDIM | výstupní produkt |
| Width | REAL | |
| Height | REAL | |
| Thickness | REAL | |
| CompAmountOut | COMPAMOUNT | počet výstp. produk. |
| Soll | UINT | - žádaný |
| Ist | UINT | - aktuální[2] |
| CompAmountIn | COMPAMOUNT | počet vstup. produk. |
| Soll | UINT | |
| Ist | UINT | |

[1] 16-ti bitová proměnná svazující příslušné funkční diagramy řídícího systému a programovatelného automatu.
[2] zpětná vazba od strojů k řídícímu systému

*4.3.1.1.2.2 .Params.SPP (Spine Preparation - opracování hřbetu)*

| | | |
|---|---|---|
| OPC_Bind | OPC_BIND | |
| General | MACHINE_GENERAL | |
| Params | BINDINGPARAMS | |
| CompDimIn | COMPDIM | |
| CompDimOut | COMPDIM | |
| CompAmountOut | COMPAMOUNT | |
| CompAmountIn | COMPAMOUNT | |
| ESGP | ENDSHEETGLUINGPARAMS | |
| CAP | COVERAPPLICATIONPARAMS | |
| SPP | SPINEPREPARATIONPARAMS | oprac. hřbetu |
| MillingDepth | REAL | - hloubka fréz. |
| NotchingDistance | REAL | - vdálen. vrubů |
| NotchingDepth | REAL | - hloubka vrubů. |

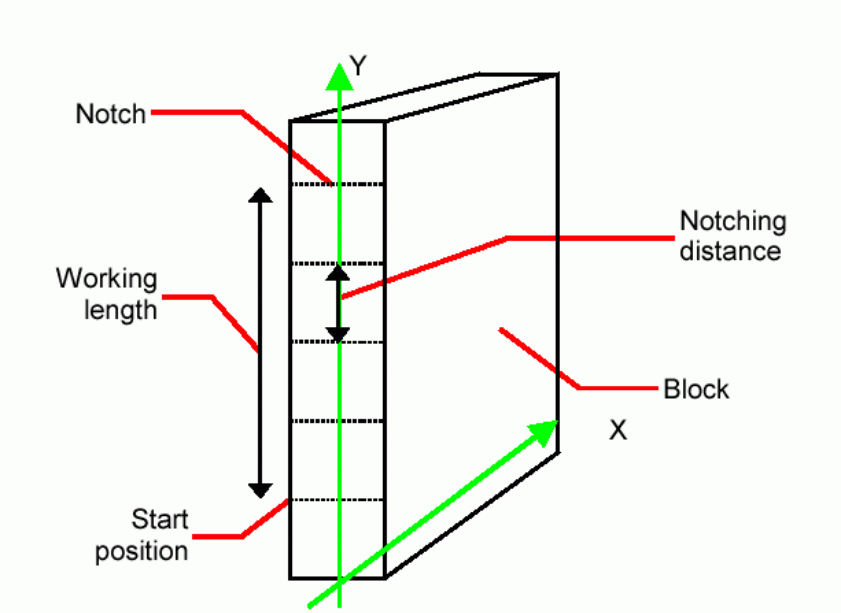



*Figure 7.20 Parameters and coordinate systems for the SpinePreparation process*

*Obr. 4.3-1 Parametry procesu obracování hřbetu*

*4.3.1.1.2.3 .Params.ESGP (ESGP – EndSheetGluing – lepení předsádky)*

| | | |
|---|---|---|
| OPC_Bind | OPC_BIND | |
| General | MACHINE_GENERAL | |
| Params | BINDINGPARAMS | |
| CompDimIn | COMPDIM | |
| CompDimOut | COMPDIM | |
| CompAmountOut | COMPAMOUNT | |
| CompAmountIn | COMPAMOUNT | |
| ESGP | ENDSHEETGLUINGPARAMS | Lepení |
| GlueBrand | STRING(10) | |
| GlueType | STRING(10) | |
| F | GLUINGDIM | F = Front (Přední) |
| WorkingPath | XYDIM | - prac. cesta |
| x | REAL | x osa |
| y | REAL | y osa |
| Offset | XYDIM | - Offset |
| x | REAL | |
| y | REAL | |
| StartPosition | XYDIM | - počát. pozice |
| x | REAL | |
| y | REAL | |
| GlueLineWidth | REAL | - šířka linie lepidla |
| B | GLUINGDIM | B = Back |
| MeltingTemperature | USINT | tavící teplota |

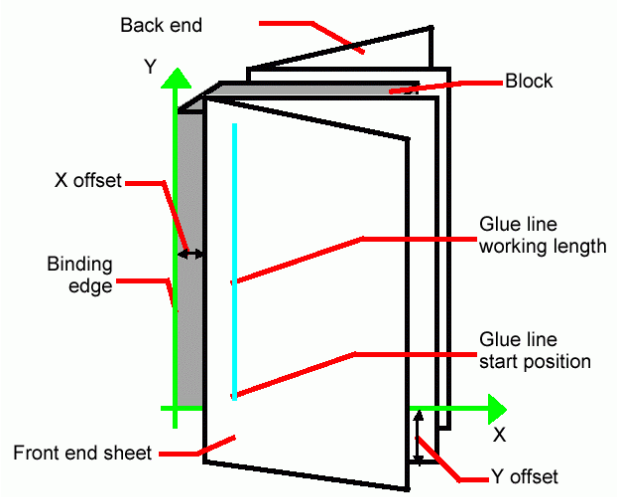



*Figure 7.9 Parameters and coordinate system used for end-sheet gluing*

*Obr. 4.3-2 Parametry lepení předsádky*

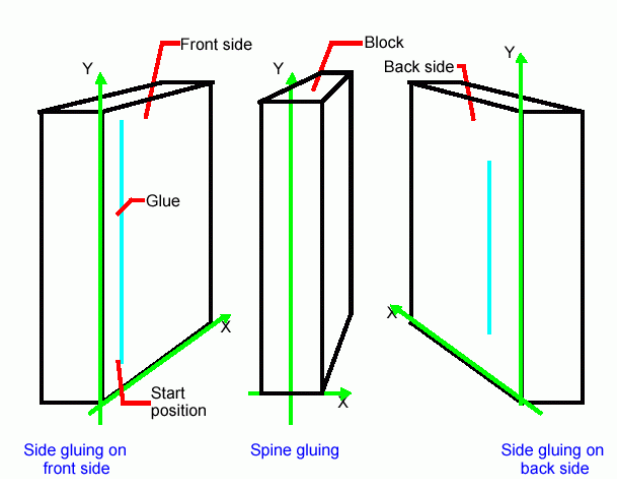



*Figure 7.14 Parameters and coordinate system for glue application*

*Obr. 4.3-3 Parametry procesu lepení*

*4.3.1.1.2.4 .Params.STP (SpineTaping – lepení stripu)*

| | |
|---|---|
| OPC_Bind | OPC_BIND |
| General | MACHINE_GENERAL |
| Params | BINDINGPARAMS |
| CompDimIn | COMPDIM |
| CompDimOut | COMPDIM |
| CompAmountOut | COMPAMOUNT |
| CompAmountIn | COMPAMOUNT |
| ESGP | ENDSHEETGLUINGPARAMS |
| CAP | COVERAPPLICATIONPARAMS |
| SPP | SPINEPREPARATIONPARAMS |
| STP | SPINETAPINGPARAMS |
| HorizontalExcess | REAL |
| TopExcess | REAL |
| StripLength | REAL |
| StripBrand | STRING(10) |
| StripColor | STRING(10) |
| StripMaterial | STRING(10) |
| Prozesse | UINT |

| |
|---|
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |

[1]

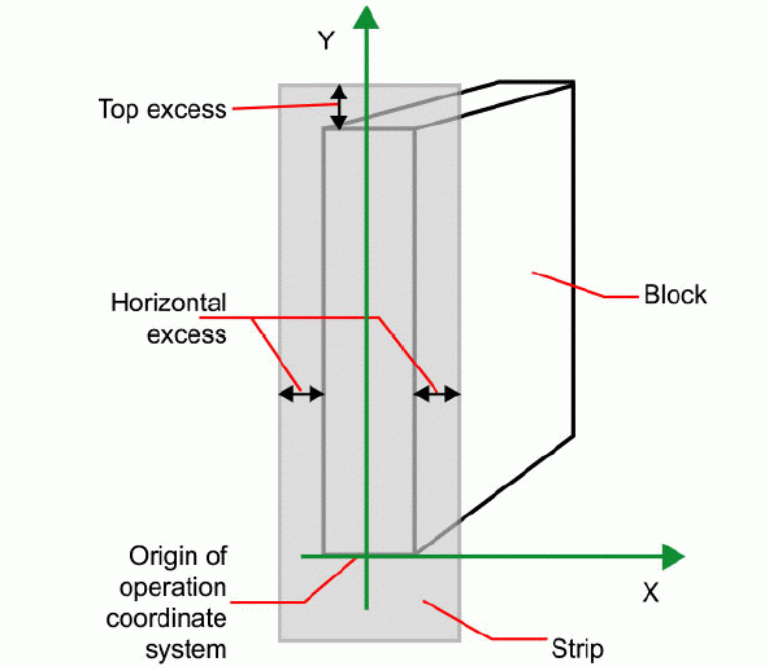



*Figure 7.21 Parameters and coordinate system for the SpineTaping process*

*Obr. 4.3-4 Parametry procesu lepení gázu*

[1] proměnná obsahuje, které procesy mají být prováděny

*4.3.1.1.2.5 .Params.CAP (CAP – Cover Application)*

| | |
|---|---|
| OPC_Bind | OPC_BIND |
| General | MACHINE_GENERAL |
| Params | BINDINGPARAMS |
| CompDimIn | COMPDIM |
| CompDimOut | COMPDIM |
| CompAmountOut | COMPAMOUNT |
| CompAmountIn | COMPAMOUNT |
| ESGP | ENDSHEETGLUINGPARAMS |
| CAP | COVERAPPLICATIONPARAMS |
| GlueBrand | STRING(10) |
| GlueType | STRING(10) |
| MeltingTemperature | USINT |
| F | GLUINGDIM |
| WorkingPath | XYDIM |
| Offset | XYDIM |
| StartPosition | XYDIM |
| GlueLineWidth | REAL |
| B | GLUINGDIM |
| SP | GLUINGDIM |
| CoverOffset | XYDIM |
| x | REAL |
| y | REAL |
| CoverDim | COMPDIM |
| Width | REAL |
| Height | REAL |
| Thickness | REAL |

F = Front

B = Back

spine = hřbet

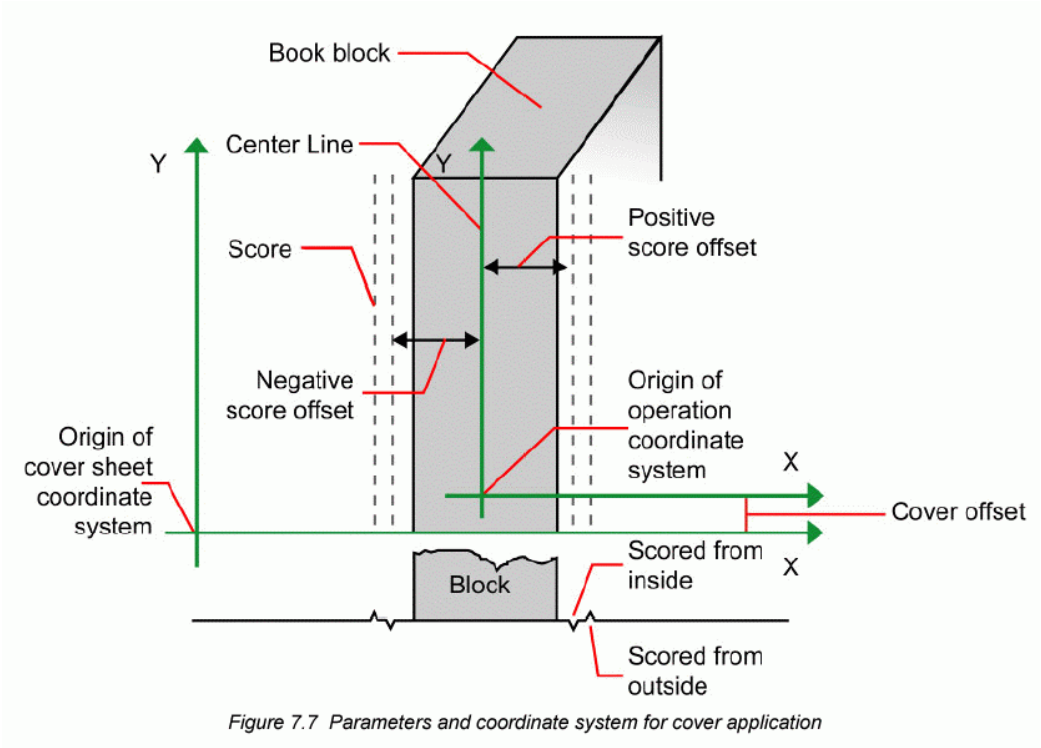



*Figure 7.7 Parameters and coordinate system for cover application*

*Obr. 4.3-5 Parametry procesu přilepení obalu*

### *4.3.1.2 OPC_Trim (řezací stroj)*

4.3.1.2.1 Params.

| | |
|---|---|
| OPC_Trim | OPC_TRIM |
| General | MACHINE_GENERAL |
| Params | TRIMPARAMS |
| CompDimIn | COMPDIM |
| Width | REAL |
| Height | REAL |
| Thickness | REAL |
| CompDimOut | COMPDIM |
| Width | REAL |
| Height | REAL |
| Thickness | REAL |
| CompAmountIn | COMPAMOUNT |
| Soll | UINT |
| Ist | UINT |
| CompAmountOut | COMPAMOUNT |
| Soll | UINT |
| Ist | UINT |
| TrimmingOffset | REAL |
| Width | REAL |
| Height | REAL |

| |
|---|
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| šířka ořezu |
| šířka oříznutého produktu |
| výška oříznutého produktu |

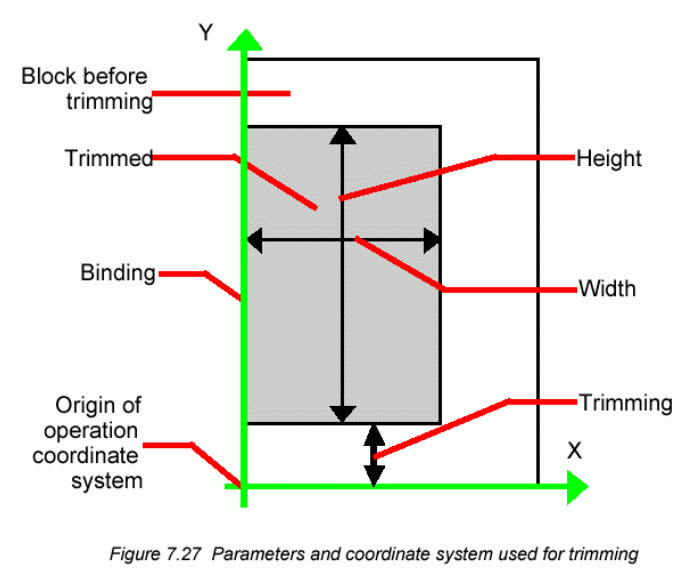


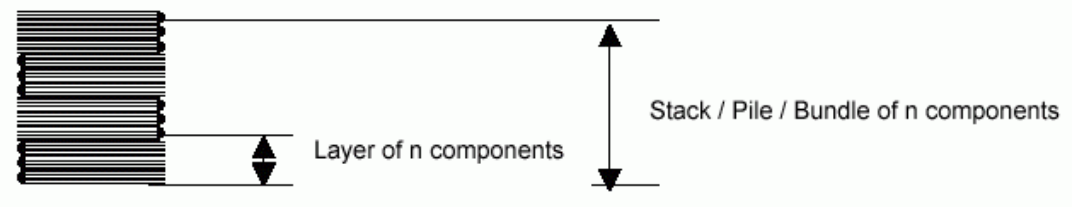

*Figure 7.27 Parameters and coordinate system used for trimming*

*Obr. 4.3-6 Parametry procesu ořezání*

### *4.3.1.3 OPC;_Stack (balící stroj)*

4.3.1.3.1 .Params

| | | |
|---|---|---|
| OPC_Stack | OPC_STACK | |
| General | MACHINE_GEN | |
| Params | STACKPARAMS | |
| CompDimIn | COMPDIM | |
| CompDimOut | COMPDIM | |
| CompAmountIn | COMPAMOUNT | |
| CompAmountOut | COMPAMOUNT | |
| StandardAmount | USINT | počet knih v balíku |
| LayerAmount | USINT | počet knih ve vrstvě |
| MinAmount | USINT | minimální počet knih ve vrstvě |
| MaxAmount | USINT | max. počet knih v balíku |
| MaxWeight | REAL | max. hmotnost balíku |
| Offset | BOOL | přesah |
| Compensate | BOOL | kompenzace |




*Obr. 4.3-7 Schéma balíku knih*

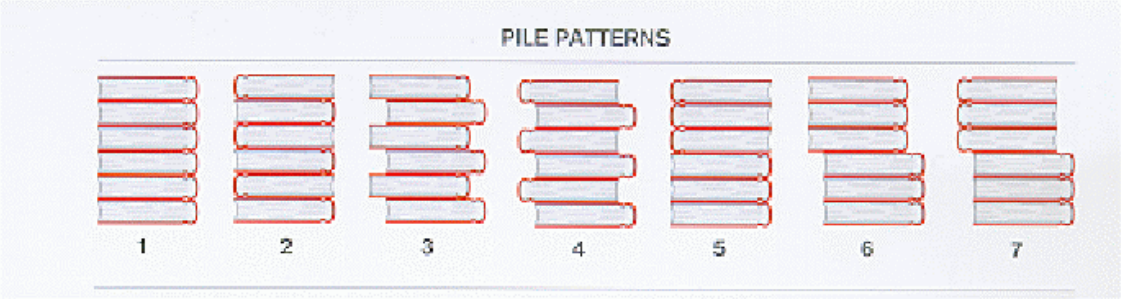



*Table 7-3 Parameters in Stacking*

| Pile Pattern | StandardAmount | LayerAmount (Default = StandardAmount) | Compensate (Default = true) | Offset (Default = false) |
|---|---|---|---|---|
| 1 | 6 | 6 | *true* | *false* |
| 2 | 6 | 1 | *true* | *false* |
| 3 | 6 | 1 | *false* | *true* |
| 4 | 6 | 1 | *true* | *true* |
| 5 | 6 | 3 | *true* | *false* |
| 6 | 6 | 3 | *false* | *true* |
| 7 | 6 | 3 | *true* | *true* |

*Obr. 4.3-8 Varianty balení knih*

### 4.3.2 Programová realizace funkčního diagramu

Na příkladu funkčního diagramu stroje “Stack” je uvedena realizace funkčního diagramu dle obr. 4.2-16 v jazyce B&R Automation Basic.

```
Select SchrittNr
                  ;0
      state UEBERWACHUNG_FREIGABE_EINTAKTUNG
            if EDGEPOS(produktion_freigabe) then
                  produktion  = 1
            endif
            when Bit_Tst(OPC_Stack.General.ControlStation.Commands,
                                    BIT_FreigabeEintaktung) = 0
                  OPC_Stack.General.MachineState.EintaktungEin = 0
                        OPC_Stack.Params.CompAmountOut.Ist = 0
                        ;Machine leeren
                        produktion  = 0
                        next WARTEN_AUF_NEW_JOB
                  ;1
      state WARTEN_AUF_NEW_JOB
            when Bit_Tst(OPC_Stack.General.ControlStation.Commands,BIT_NewJob)
                   ;read the data
                   OPC_Stack_Active = OPC_Stack
                   next CHECK_DATA
                  ;2
      state CHECK_DATA
            ;check data
            OPC_Stack.General.MachineState.DataOK = 1
            when OPC_Stack.General.MachineState.DataOK = 1
                  OPC_Stack.General.MachineState.EinrichtenOK = 0
                  OPC_Stack.General.ControlStation.Commands = \\
            Bit_Set(OPC_Stack.General.ControlStation.Commands,BIT_DataOK)
                  next EINRICHTEN
                  when OPC_Stack.General.MachineState.DataOK = 0
                        OPC_Stack.General.ControlStation.Commands = \\
      Bit_Set(OPC_Stack.General.ControlStation.Commands,BIT_DataIncorrect)
                  next DATA_ERROR
                  ;3
      state EINRICHTEN
            ;machine einrichten
            when 1=1
                  next WARTEN_MASCHINE_BEREIT
                  ;4
      state WARTEN_MASCHINE_BEREIT
            when OPC_Stack.General.MachineState.MaschineBereit = 0
                  OPC_Stack.General.ControlStation.Commands = \\
      Bit_Set(OPC_Stack.General.ControlStation.Commands,BIT_MachineReady)
                  next WARTEN_NEW_JOB_RUECKGESETZT
                  ;5
      state WARTEN_NEW_JOB_RUECKGESETZT
            when Bit_Tst(OPC_Stack.General.ControlStation.Commands,BIT_NewJob)
= 0
                  OPC_Stack.General.ControlStation.Commands = \\
            Bit_Clr(OPC_Stack.General.ControlStation.Commands,BIT_DataOK)
                        OPC_Stack.Params.CompAmountOut.Ist = 0
                  next WARTEN_EINTAKTUNG_FREIGEGEBEN
                  ;6
      state WARTEN_EINTAKTUNG_FREIGEGEBEN
            when
Bit_Tst(OPC_Stack.General.ControlStation.Commands,BIT_FreigabeEintaktung) = 1
                  next UEBERWACHUNG_FREIGABE_EINTAKTUNG
                  ;7
      state DATA_ERROR
            when 1=1
            next UEBERWACHUNG_FREIGABE_EINTAKTUNG
endselect
```

# 5 Závěr

Výsledek této práce zabývající se problematikou plně automatické linky pro produkci malonákladových knih představuje funkční soubor řešení pro řízení výrobní linky nadřazeným systémem a komunikaci s podřízenými programovatelnými automaty.

Návrhy, postupy a řešení realizované v rámci této práce jsou v mnoha ohledech v daném oboru nové a jejich nasazení umožní zvýšení efektivity výroby.

Na druhou stranu je ale dlužno říci, že představené řešení je pouze prvním prototypem, který dává odpověď na principielní otázky způsobu realizace. Jeho nasazení v reálných podmínkách u zákazníka – průmyslové knihárny není zatím možné. Pro reálné nasazení je nutné doplnit systém o další funkce a vlastnosti, jejichž realizace z hlediska časové náročnosti dalece přesahovala možnosti této práce.

Uvedené řešení ale přesto představuje solidní základ pro další vývoj řídicího systému výrobní linky. Způsob realizace řídicího systému je principielně vyřešen a zbývá doplnění rozšiřujících funkcí.

Navržené řešení je univerzální z hlediska konfigurace linky. To znamená, že je možno konfigurovat počet strojů v lince stejně tak jako spektrum procesů, které mají být na strojích prováděny. Konkrétní popis je pak uveden na příkladu konfigurace linky sestávající se ze tří strojů – lepícího a vázacího stroje, ořezávacího stroje a balícího stroje. V době vypracování této práce nebyly bohužel dostupné všechny stroje alespoň pro tuto minimální konfiguraci linky, ale funkčnost řešení byla s úspěchem vyzkoušena na simulované výrobě pomocí programovatelného automatu.

# 6 Literatura

## *6.1 Knihy a časopisy*


[1] Zahrádka, J.: Dokončovací výroba pro 3.ročník SPŠG, Státní pedagogické nakladatelství, Praha, 1988

[2] Liebau, D - Heinze, I: Industrielle Buchbinderei, Verlag Beruf + Schule, Itzehoe, 1997.

[3] Beneš,P. - Chlebný,J - Langer, J - Martinásková, M - Voráček, V.: Automatizace a automatizační technika III, Prostředky automatického řízení. Computer Press, vydání první, 2000

[4] Hlava,J. - Prostředky automatického řízení II, Vydavatelství ČVUT, Praha, 2000

[5] Kriesel,W. - Heimbold, T - Telschow,D.: Bustechnologien für die Automation, Zusatz z. Titel: Vernetzung, Auswahl und Anwendung von Kommunikationssystemen, Hüthig, Heidelberg, 2. Ausgabe, 2000, ISBN/ISSN 3-7785-2778-9

[6] Haasz V. Průmyslové komunikační systémy, Automatizace 40 (1997) č.9 str. 540-544

[7] Daďo S. HART – protokol komunikace inteligentních senzorů, Automatizace 40 (1997) č.9 str. 545-549

[8] Bartůněk I. Distribuované moduly CAN pro průmyslové aplikace, Automatizace 39 (1996) č.9 str. 423-424

[9] Bartůněk I. CAN – Controller – Area - Network Automatizace 39 (1996) č.4 str. 159-163

[10] Klüger P. – Dehof, M.: Das Security-Problem, Computer & Automation 11-2003, str. 44 - 47, 2003, Weka Fachzeitschriften Verlag

[11] Dobezernz, W; Kowalski, T – Visual Basic 4, München; Wien, Hanse, 1996


## *6.2 Internetové odkazy*


[www1] http://fieldbus.feld.cvut.cz - *Materiály sdružení Fieldbus, přehled mnoha sběrnic*

[www2] http://www.br-automation.com - *Firemní materiály firmy B&R*

[www3] http://golis.uninet.cz - *Golisovy programovací stránky, část čárové kódy*

[www4] http://www.cip4.org/ - *oficiální stránka organizace CIP4*

[www5] http://www.w3.org/XML/Schema

[www6] http://www.iconics.com - *oficiální stránka firmy Iconics, vyrábějící produkt Genesis*

[www7] http://www.iconics.co.uk - *britská stránka firmy Iconics, zde je možné se zdarma registrovat a získat tak přístup do „knowledge base" – stránek obsahujících četné příklady a návody především na použití VBA v rámci produktu Genesis*

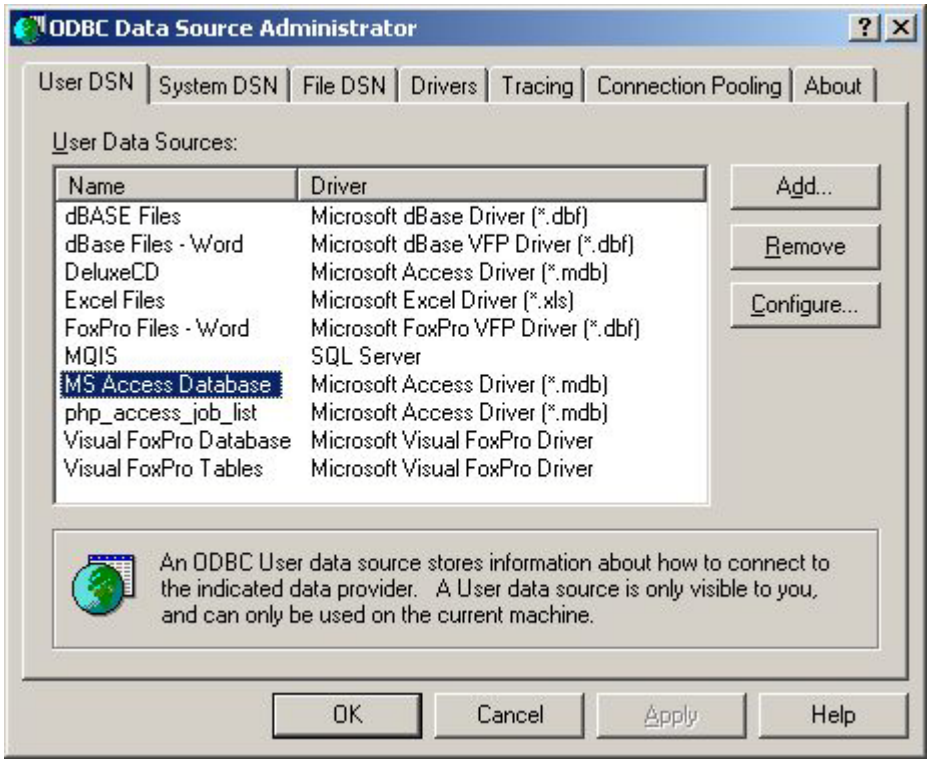
[www8] http://www.iainsider.co.uk/scadasites.htm - *stránka obsahující velké množství abecedně setříděných odkazů z oblasti automatizace především z pohledu SCADA systémů*

# 7 Přílohy

## *7.1 Návody*

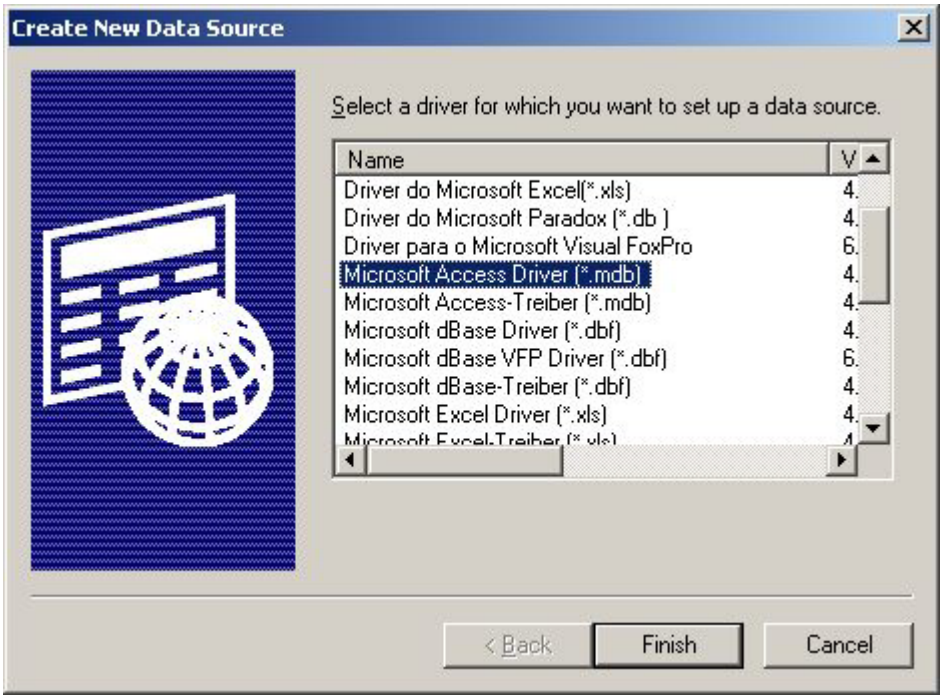
### 7.1.1 Vytvoření DNS zdroje využívajícího ODBC drivery

*1)* Otevřeme ovládací panel (*Menu Start -> Nastavení (Settings)-> Ovládací panel - Control Panel)*
2) Vybereme položku „Administrátorské nástroje“ (Administrative Tools) a otevřeme ji.
3) Vybereme položku „Data Sources“ a otevřeme ji. Objeví se dialogové okno podobné jako na obr. 7.1-1.



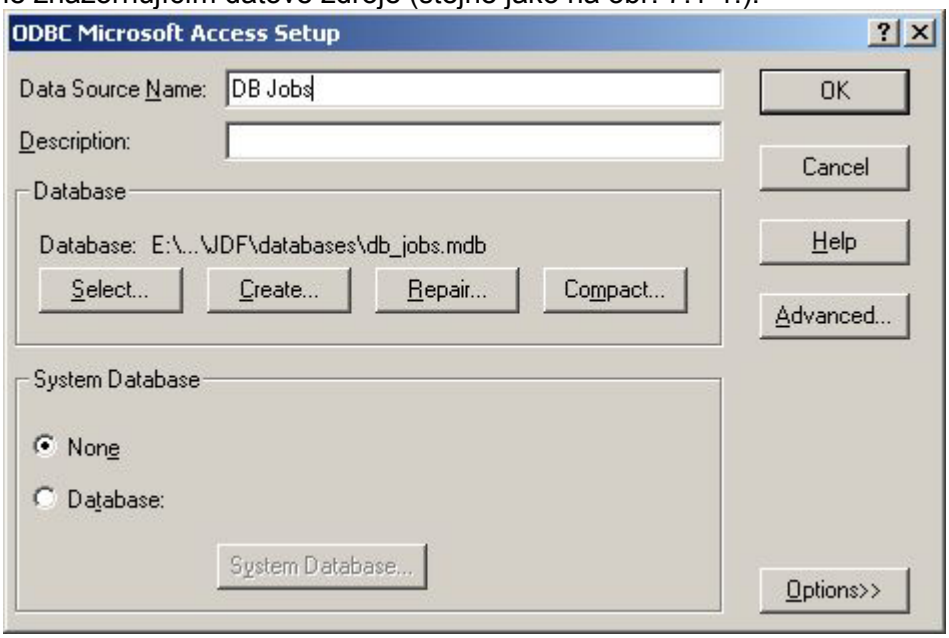

*Obr. 7.1-1 Dialogové okno ODBC datové zdroje (Data Sources)*

4) V dialogovém okně vybereme driver Microsoft Access Driver a pak stiskneme tlačítko přidat („Add“) v záložce User DNS (viz obr. 7.1-2)




*Obr. 7.1-2 Dialogové okno pro vytvoření nového ODBC zdroje*

5) Objeví se dialogové okno podobné jako na obr. 7.1-3. Zde je nutné vyplnit název nového zdroje a rovněž vybrat databázi, ke které se má pomocí ODBC přistupovat. Po stisknutí tlačítka OK pak dojde k vytvoření nového zdroje, který se zobrazí v aktualizovaném dialogovém okně znázorňujícím datové zdroje (stejné jako na obr. 7.1-1.).




*Obr. 7.1-3 Dialogové okno pro nastavení ODBC zdroje pro přístup k MS Access*

## *7.2 JDF*

### 7.2.1 JDF schéma - ukázka části Trimming

```
      <!--TrimmingParams-->
      <xs:element name="TrimmingParams" type="jdf:TrimmingParams_r"
substitutionGroup="jdf:Resource"/>
      <xs:element name="TrimmingParamsLink" type="jdf:ParameterLink"
substitutionGroup="jdf:ResourceLink"/>
      <xs:attributeGroup name="TrimmingParamsAttribs_c">
            <xs:attribute name="Width" type="xs:double" use="optional"/>
            <xs:attribute name="Height" type="xs:double" use="optional"/>
```

```
            <xs:attribute name="TrimmingOffset" type="xs:double"
use="optional"/>
      </xs:attributeGroup>
      <xs:attributeGroup name="TrimmingParamsAttribs_u">
            <xs:attributeGroup ref="jdf:TrimmingParamsAttribs_c"/>
            <xs:attribute name="TrimmingType" use="optional">
                  <xs:simpleType>
                        <xs:restriction base="xs:NMTOKEN">
                              <xs:enumeration value="Detailed"/>
                              <xs:enumeration value="SystemSpecified"/>
                        </xs:restriction>
                  </xs:simpleType>
            </xs:attribute>
      </xs:attributeGroup>
      <xs:attributeGroup name="TrimmingParamsAttribs_r">
            <xs:attributeGroup ref="jdf:TrimmingParamsAttribs_c"/>
            <xs:attribute name="TrimmingType" use="required">
                  <xs:simpleType>
                        <xs:restriction base="xs:NMTOKEN">
                              <xs:enumeration value="Detailed"/>
                              <xs:enumeration value="SystemSpecified"/>
                        </xs:restriction>
                  </xs:simpleType>
            </xs:attribute>
      </xs:attributeGroup>
      <xs:complexType name="TrimmingParams_r">
            <xs:complexContent>
                  <xs:extension base="jdf:ParameterResource">
                        <xs:sequence minOccurs="0" maxOccurs="unbounded">
                              <xs:group ref="jdf:GenericElements"
minOccurs="0"/>
                              <xs:element name="TrimmingParamsUpdate"
type="jdf:TrimmingParams_ru" minOccurs="0"/>
                              <xs:element name="TrimmingParams"
type="jdf:TrimmingParams_rp" minOccurs="0"/>
                        </xs:sequence>
                        <xs:attributeGroup ref="jdf:TrimmingParamsAttribs_r"/>
                  </xs:extension>
            </xs:complexContent>
      </xs:complexType>
      <xs:complexType name="TrimmingParams_re">
            <xs:complexContent>
                  <xs:extension base="jdf:ResourceElement">
                        <xs:sequence minOccurs="0" maxOccurs="unbounded">
                              <xs:group ref="jdf:GenericElements"
minOccurs="0"/>
                        </xs:sequence>
                        <xs:attributeGroup ref="jdf:TrimmingParamsAttribs_r"/>
                  </xs:extension>
            </xs:complexContent>
      </xs:complexType>
      <xs:complexType name="TrimmingParams_rp">
            <xs:complexContent>
                  <xs:extension base="jdf:ParameterResourceLeaf">
                        <xs:sequence minOccurs="0" maxOccurs="unbounded">
                              <xs:group ref="jdf:GenericElements"
minOccurs="0"/>
                              <xs:element name="TrimmingParams"
type="jdf:TrimmingParams_rp" minOccurs="0"/>
                        </xs:sequence>
```

```
                        <xs:attributeGroup ref="jdf:TrimmingParamsAttribs_u"/>
                  </xs:extension>
            </xs:complexContent>
      </xs:complexType>
      <xs:complexType name="TrimmingParams_ru">
            <xs:complexContent>
                  <xs:extension base="jdf:ResourceUpdate">
                        <xs:sequence minOccurs="0" maxOccurs="unbounded">
                              <xs:group ref="jdf:GenericElements"
minOccurs="0"/>
                        </xs:sequence>
                        <xs:attributeGroup ref="jdf:TrimmingParamsAttribs_u"/>
                  </xs:extension>
            </xs:complexContent>
      </xs:complexType>
      <xs:complexType name="TrimmingParams_rue">
            <xs:complexContent>
                  <xs:extension base="jdf:ResourceUpdateElement">
                        <xs:sequence minOccurs="0" maxOccurs="unbounded">
                              <xs:group ref="jdf:GenericElements"
minOccurs="0"/>
                        </xs:sequence>
                        <xs:attributeGroup ref="jdf:TrimmingParamsAttribs_u"/>
                  </xs:extension>
            </xs:complexContent>
      </xs:complexType>
```

### 7.2.2 Použitý JDF dokument pro práci s daty specifikujícími knihu (pro softcover s předsádkou)

```
<?xml version="1.0" encoding="UTF-8"?>
<JDF xmlns="http://www.CIP4.org/JDFSchema_1_1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" Type="ProcessGroup" ID="JSoftcover" Status="Cleanup"
xsi:schemaLocation="http://www.CIP4.org/JDFSchema_1_1 " Version="1.1">
      <NodeInfo DueLevel="Trivial" Start="2002-08-11T12:00:00+02:00" FirstStart="2002-08-
11T11:00:00+02:00" LastStart="2002-08-11T12:30:00+02:00" End="2002-08-11T14:00:00+02:00"
FirstEnd="2002-09-02T13:00:00+02:00" LastEnd="2002-08-11T15:20:00+02:00" TotalDura-
tion="P00Y00M00DT02H00M" SetupDuration="P00Y00M00DT00H15M" CleanupDuration="P00Y00M00DT00H30M"/>
      <ResourcePool>
            <FoldingParams ID="Par1" DescriptionType="FoldCatalog" Status="Available" Fold-
Catalog="F32-2" Class="Parameter"/>
            <GatheringParams ID="Par2" Status="Available" Class="Parameter">
                  <Disjointing ID="asd" Number="10" Offset="10 10" OffsetAmount="10" Offset-
Direction="Straight"/>
            </GatheringParams>
            <EndSheetGluingParams ID="Par41" Status="Available" Class="Parameter">
                  <EndSheet ID="inp161" Offset="0.000 0.000" Side="Front">
                        <GlueLine ID="inp161GL" StartPosition="0.000 0.000" Working-
Path="0.000 918.425" AreaGlue="false" GluingPattern="1 0" GlueLineWidth="28.346" Glue-
Type="ColdGlue" MeltingTemperature="0" GlueBrand="AAAx"/>
                  </EndSheet>
                  <EndSheet ID="inp162" Offset="0.000 0.000" Side="Back">
                        <GlueLine ID="inp162GL" StartPosition="0.000 0.000" Working-
Path="0.000 918.425" AreaGlue="false" GluingPattern="1 0" GlueLineWidth="28.346" Glue-
Type="ColdGlue" MeltingTemperature="0" GlueBrand="AAAx"/>
                  </EndSheet>
            </EndSheetGluingParams>
            <SpinePreparationParams ID="Par3" Status="Available" MillingDepth="5.669"
Class="Parameter" NotchingDistance="11.339" NotchingDepth="0.567"/>
            <SpineTapingParams ID="Par5" Status="Available" Class="Parameter" HorizontalEx-
cess="14.173" StripLength="918.425" TopExcess="0.000" StripBrand="BrandXYZ" StripColor="Blue"
StripMaterial="Paper">
                  <GlueApplicationRef rRef="GASpineTapingSpine"/>
                  <GlueApplicationRef rRef="GASpineTapingFront"/>
                  <GlueApplicationRef rRef="GASpineTapingBack"/>
            </SpineTapingParams>
            <CoverApplicationParams ID="Par6" CoverOffset="0.000 0.000" Status="Available"
Class="Parameter">
                  <GlueApplicationRef rRef="GACoverSpine"/>
```

```
                    <GlueApplicationRef rRef="GACoverFront"/>
                    <GlueApplicationRef rRef="GACoverBack"/>
             </CoverApplicationParams>
             <TrimmingParams ID="Par7" Status="Available" Class="Parameter" Trimming-
Type="Detailed" Width="595.276" Height="850.394" TrimmingOffset="39.685"/>
             <StackingParams ID="Par8" Class="Parameter" Status="Available" StandardAmount="4"
MaxAmount="4" MinAmount="1" MaxWeight="15000.000" Compensate="True" Offset="True" layerA-
mount="2"/>
             <Component ID="Res101" Class="Quantity" Status="Available" ComponentType="Ribbon"
IsWaste="false" ProductType="unknown" Dimensions="0 0 0">
                    <Location LocationName="Feeder-5"/>
             </Component>
             <Component ID="Res102" Class="Quantity" Status="Available" ComponentType="Ribbon"
IsWaste="false" ProductType="unknown" Dimensions="0 0 0" OverfoldSide="Front">
                    <Location LocationName="Feeder-5"/>
             </Component>
             <Component ID="Res111" Class="Quantity" Status="Available" Component-
Type="PartialProduct" IsWaste="false" ProductType="unknown" Dimensions="  ">
                    <Location LocationName="Feeder-5"/>
             </Component>
             <Component ID="Res112" Class="Quantity" Status="Available" Component-
Type="PartialProduct" IsWaste="false" ProductType="unknown" Dimensions="  ">
                    <Location LocationName="Feeder-5"/>
             </Component>
             <Component ID="Res12" Class="Quantity" Status="Available" Component-
Type="PartialProduct" IsWaste="false" ProductType="unknown" Dimensions="651.968 918.425
113.386"/>
             <Component ID="Res13" Class="Quantity" Status="Available" Component-
Type="PartialProduct" IsWaste="false" ProductType="unknown" Dimensions="646.299 918.425
113.386"/>
             <Component ID="Res132" Class="Quantity" Status="Available" ComponentType="Sheet"
IsWaste="false" ProductType="unknown" Dimensions="651.968 918.425 0.283"/>
             <Component ID="Res133" Class="Quantity" Status="Available" ComponentType="Sheet"
IsWaste="false" ProductType="unknown" Dimensions="651.968 918.425 0.283"/>
             <Component ID="Res14" Class="Quantity" Status="Available" Component-
Type="PartialProduct" IsWaste="false" ProductType="unknown" Dimensions="651.968 918.425
113.952"/>
             <Component ID="Res15" Class="Quantity" Status="Available" Component-
Type="PartialProduct" IsWaste="false" ProductType="BookBlock" Dimensions="651.968 918.425
113.952"/>
             <Component ID="Res152" Class="Quantity" Status="Available" ComponentType="Sheet"
IsWaste="false" ProductType="Cover" Dimensions="651.968 918.425 2.835"/>
             <Component ID="Res16" Class="Quantity" Status="Available" ComponentType="Block"
IsWaste="false" ProductType="unknown" Dimensions="654.803 918.425 119.622"/>
             <Component ID="Res17" Class="Quantity" Status="Available" Component-
Type="FinalProduct" IsWaste="false" ProductType="Book" Dimensions="595.276 850.394 119.622"/>
             <Component ID="Res18" Class="Quantity" Status="Available" Component-
Type="PartialProduct" IsWaste="false" ProductType="unknown" Dimensions="595.276 850.394 478.488">
                    <Bundle BundleType="Stack" TotalAmount="50">
                           <BundleItem Amount="50">
                                  <ComponentRef rRef="Res17"/>
                           </BundleItem>
                    </Bundle>
             </Component>
             <Media ID="Med1" Status="Available" Class="Consumable" Dimension="6236.220
9070.866" MediaType="Paper" Amount="10000"/>
             <GluingParams ID="Par4" Status="Available" Class="Parameter">
                    <Glue WorkingDirection="Top">
                           <GlueApplication ID="GACoverSpine" GluingTechnique="SpineGluing">
                                  <GlueLineRef rRef="GLCoverSpine"/>
                           </GlueApplication>
                    </Glue>
                    <Glue WorkingDirection="Top">
                           <GlueApplication ID="GACoverFront" GluingTech-
nique="SideGluingFront">
                                  <GlueLineRef rRef="GLCoverFront"/>
                           </GlueApplication>
                    </Glue>
                    <Glue WorkingDirection="Top">
                           <GlueApplication ID="GACoverBack" GluingTechnique="SideGluingBack">
                                  <GlueLineRef rRef="GLCoverBack"/>
                           </GlueApplication>
                    </Glue>
                    <Glue WorkingDirection="Top">
                           <GlueApplication ID="GASpineTapingSpine" GluingTech-
nique="SpineGluing">
```

```
                                <GlueLineRef rRef="GLSpineTapingSpine"/>
                        </GlueApplication>
                </Glue>
                <Glue WorkingDirection="Top">
                        <GlueApplication ID="GASpineTapingFront" GluingTech-
nique="SideGluingFront">
                                <GlueLineRef rRef="GLSpineTapingFront"/>
                        </GlueApplication>
                </Glue>
                <Glue WorkingDirection="Top">
                        <GlueApplication ID="GASpineTapingBack" GluingTech-
nique="SideGluingBack">
                                <GlueLineRef rRef="GLSpineTapingBack"/>
                        </GlueApplication>
                </Glue>
        </GluingParams>
        <GlueLine ID="GLSpineTapingSpine" Status="Available" Class="Parameter" Working-
Path="0 918.425" StartPosition="0 0" AreaGlue="false" GluingPattern="1 0" GlueLineWidth="28.346"
GlueType="Hotmelt" GlueBrand="BBBx" MeltingTemperature="200"/>
        <GlueLine ID="GLSpineTapingFront" Status="Available" Class="Parameter" Working-
Path="0 918.425" StartPosition="0 0" AreaGlue="false" GluingPattern="1 0" GlueLineWidth="28.346"
GlueType="Hotmelt" GlueBrand="BBBx" MeltingTemperature="200"/>
        <GlueLine ID="GLSpineTapingBack" Status="Available" Class="Parameter" Working-
Path="0 918.425" StartPosition="0 0" AreaGlue="false" GluingPattern="1 0" GlueLineWidth="28.346"
GlueType="Hotmelt" GlueBrand="BBBx" MeltingTemperature="200"/>
        <GlueLine ID="GLCoverSpine" Status="Available" Class="Parameter" Working-
Path="0.000 918.425" StartPosition="0.0 0.0" AreaGlue="false" GluingPattern="1 0"
GlueLineWidth="28.346" GlueType="ColdGlue" GlueBrand="" MeltingTemperature="0"/>
        <GlueLine ID="GLCoverFront" Status="Available" Class="Parameter" Working-
Path="0.000 918.425" StartPosition="0.000 0.000" AreaGlue="false" GluingPattern="1 0"
GlueLineWidth="28.346" GlueType="ColdGlue" GlueBrand="BBBx" MeltingTemperature="0"/>
        <GlueLine ID="GLCoverBack" Status="Available" Class="Parameter" WorkingPath="0.000
918.425" StartPosition="0.000 0.000" AreaGlue="false" GluingPattern="1 0" GlueLineWidth="28.346"
GlueType="ColdGlue" GlueBrand="" MeltingTemperature="0"/>
    </ResourcePool>
    <JDF ID="J111" Status="Waiting" Type="Folding" JobID="SRSSoftcover">
        <ResourceLinkPool>
            <FoldingParamsLink Usage="Input" rRef="Par1"/>
            <ComponentLink Usage="Input" rRef="Res101" Amount="20"/>
            <ComponentLink Usage="Output" rRef="Res111" Amount="20"/>
        </ResourceLinkPool>
    </JDF>
    <JDF ID="J12" Status="Waiting" Type="Gathering" JobID="SRSSoftcover">
        <ResourceLinkPool>
            <GatheringParamsLink Usage="Input" rRef="Par2"/>
            <ComponentLink Usage="Input" rRef="Res111" Amount="20"/>
            <ComponentLink Usage="Input" rRef="Res112" Amount="20"/>
            <ComponentLink Usage="Output" rRef="Res12" Amount="20"/>
        </ResourceLinkPool>
    </JDF>
    <JDF ID="J13" Status="Waiting" Type="SpinePreparation" JobID="SRSSoftcover">
        <ResourceLinkPool>
            <SpinePreparationParamsLink Usage="Input" rRef="Par3"/>
            <ComponentLink Usage="Input" rRef="Res12" Amount="20"/>
            <ComponentLink Usage="Output" rRef="Res13" Amount="20"/>
        </ResourceLinkPool>
    </JDF>
    <JDF ID="J14" Status="Waiting" Type="EndSheetGluing" JobID="SRSSoftcover">
        <ResourceLinkPool>
            <EndSheetGluingParamsLink Usage="Input" rRef="Par41"/>
            <ComponentLink Usage="Input" rRef="Res13" Amount="20"/>
            <ComponentLink Usage="Input" rRef="Res132" Amount="20"/>
            <ComponentLink Usage="Input" rRef="Res133" Amount="20"/>
            <ComponentLink Usage="Output" rRef="Res14" Amount="20"/>
        </ResourceLinkPool>
    </JDF>
    <JDF ID="J15" Status="Waiting" Type="SpineTaping" JobID="SRSSoftcover">
        <ResourceLinkPool>
            <SpineTapingParamsLink Usage="Input" rRef="Par5"/>
            <ComponentLink Usage="Input" rRef="Res14" Amount="20"/>
            <ComponentLink Usage="Output" rRef="Res15" Amount="20"/>
        </ResourceLinkPool>
    </JDF>
    <JDF ID="J16" Status="Waiting" Type="CoverApplication" JobID="SRSSoftcover">
        <ResourceLinkPool>
```

```
                    <CoverApplicationParamsLink Usage="Input" rRef="Par6"/>
                    <ComponentLink Usage="Input" rRef="Res15" Amount="20"/>
                    <ComponentLink Usage="Input" rRef="Res152" Amount="20"/>
                    <ComponentLink Usage="Output" rRef="Res16" Amount="20"/>
             </ResourceLinkPool>
      </JDF>
      <JDF ID="J17" Status="Waiting" Type="Trimming" JobID="SRSSoftcover">
             <ResourceLinkPool>
                    <TrimmingParamsLink Usage="Input" rRef="Par7"/>
                    <ComponentLink Usage="Input" rRef="Res16" Amount="20"/>
                    <ComponentLink Usage="Output" rRef="Res17" Amount="20"/>
             </ResourceLinkPool>
      </JDF>
      <JDF ID="J18" Status="Waiting" Type="Stacking" JobID="SRSSoftcover" JobPartID="Stack"
Activation="Active">
             <ResourceLinkPool>
                    <StackingParamsLink Usage="Input" rRef="Par8"/>
                    <ComponentLink Usage="Input" rRef="Res17" Amount="20"/>
                    <ComponentLink Usage="Output" rRef="Res18" Amount="20"/>
             </ResourceLinkPool>
      </JDF>
      <CustomerInfo CustomerJobName="Softcover" CustomerID="ID123" DescriptiveName="Softcover1">
             <Contact ContactTypes="Customer">
                    <Company OrganizationName="SRS"/>
             </Contact>
      </CustomerInfo>
</JDF>
```

## 7.3 Některé zajímavé části programového kódu

### 7.3.1 VBA

#### 7.3.1.1 Procedura pro konfiguraci strojů (inicializaci objektů)

```
Private Sub ConfigurateMachines()
    Dim i As Integer, j As Integer
    Dim s As String
    Dim MachineNr As Integer
    Dim ConfigFile As String
    ConfigFile = PATH_TO_CONFIG_FILE + "machines_config.cfg"
    Dim MachinesConfigArray() As String
    Dim MachinesConfigArray2() As String

    Call ReadConfigFile(ConfigFile, MachinesConfigArray)
    '    ReDim MachinesConfigArray(5, 2) As String
        'ReDim MachinesConfigArray(CONFIG_PARAMS_MAX,
UBound(MachinesConfigArray)) As String
    'MachinesConfigArray = MachinesConfigArray2
    TotalNrOfConfiguratedMachines = 0
    For i = 0 To UBound(MachinesConfigArray, 2)
        'MachineNr;Machine_Type;Process1;Process2;Process3;Process4
        '0        ;1           ;2       ;3       ;4       ;5
        If MachinesConfigArray(0, i) <> "#" Then
            MachineNr = CInt(MachinesConfigArray(0, i))
            If MachineNr <> i Then
                'Not valid format
                MsgBox "Sorry, but the config file is not valid"
                GoTo Error_in_config_file
            End If
            Set Machines(MachineNr) = New MachineClass

            Select Case MachinesConfigArray(1, i)   'Machine_Type

                Case "Bind":
```

```
                    TotalNrOfConfiguratedMachines =
TotalNrOfConfiguratedMachines + 1
                    'MachineNr = CInt(MachinesConfigArray(0, i))
                    'Set Machines(MachineNr) = New MachineClass
            '***Bind **************
                    With Machines(MachineNr)
                        .Name = "Bind"
                        '.JDFName = "ESGP"   'Temp
                        For j = 0 To PROCESS_NAMES_MAX - 1
                            Select Case MachinesConfigArray(j + 2, i)
                                Case "SpinePreparation"
                                    Call .SetJDFProcessName(j, "SPP")
                                Case "EndSheetGluing"
                                    Call .SetJDFProcessName(j, "ESGP")
                                Case "CoverApplication"
                                    Call .SetJDFProcessName(j, "CAP")
                                Case "SpineTaping"
                                    Call .SetJDFProcessName(j, "STP")
                                Case Else
                                    Call .SetJDFProcessName(j, "")
                            End Select
                        Next j
                        .InitializeJobs
                    End With
            '***Dry**************
                Case "Dry":
                    TotalNrOfConfiguratedMachines =
TotalNrOfConfiguratedMachines + 1
                    'MachineNr = CInt(MachinesConfigArray(0, i))
                    'Set Machines(MachineNr) = New MachineClass
                    With Machines(MachineNr)
                        .Name = "Dry"
                        For j = 0 To PROCESS_NAMES_MAX - 1
                            Select Case MachinesConfigArray(j + 2, i)
                                Case "Drying"
                                    Call .SetJDFProcessName(j, "")
                                Case Else
                                    Call .SetJDFProcessName(j, "")
                            End Select
                        Next j
                        .InitializeJobs
                    End With
            '***Trim**************
                Case "Trim":
                    TotalNrOfConfiguratedMachines =
TotalNrOfConfiguratedMachines + 1
                    'MachineNr = CInt(MachinesConfigArray(0, i))
                    'Set Machines(MachineNr) = New MachineClass
                    With Machines(MachineNr)
                        .Name = "Trim"
                        For j = 0 To PROCESS_NAMES_MAX - 1
                            Select Case MachinesConfigArray(j + 2, i)
                                Case "Trimming"
                                    Call .SetJDFProcessName(j, "TrP")
                                Case Else
                                    Call .SetJDFProcessName(j, "")
                            End Select
                        Next j
                        .InitializeJobs
                    End With
```

```
            '***Stack**************
                Case "Stack":
                    TotalNrOfConfiguratedMachines =
TotalNrOfConfiguratedMachines + 1
                    With Machines(MachineNr)
                        .Name = "Stack"
                        For j = 0 To PROCESS_NAMES_MAX - 1
                            Select Case MachinesConfigArray(j + 2, i)
                                Case "Stacking"
                                    Call .SetJDFProcessName(j, "StP")
                                Case Else
                                    Call .SetJDFProcessName(j, "")
                            End Select
                        Next j
                        .InitializeJobs
                    End With
            End Select
        End If
    Next i
    DisplayMachinesConfiguration (TotalNrOfConfiguratedMachines)
    Exit Sub
Error_in_config_file:
    'Initialize if error with default (null) values
    For i = 0 To JOBSMAX - 1
        Set Machines(i) = New MachineClass
        With Machines(i)
            .Name = ""
            For j = 0 To PROCESS_NAMES_MAX - 1
                Call .SetJDFProcessName(j, "")
            Next j
            .InitializeJobs
        End With
    Next i
    DisplayMachinesConfiguration (TotalNrOfConfiguratedMachines)
    'DisplayMachinesConfiguration (UBound(MachinesConfigArray, 2) + 1)

End Sub
```

#### *7.3.1.2 Načtení tabulky databáze Access do displeje GraphWorX*

```
Public Sub UpdateDB(Optional CalledFrom As String)
    '*********** Precedure that updates the table from Database *************

'Public Sub UpdateDB(Optional OrderByFieldNr As Integer)


    Dim QueryString, OrderByFieldName As String
    Dim temp
    Dim OrderAscending As Boolean
    'If IsMissing(OrderByFieldNr) Then OrderByFieldNr = 0

    OrderByFieldNr = GetValue("~~OrderByFieldNr~~")

    'MsgBox (OrderByFieldNr)
    DBJobsPreparedFields = Array("file_name", "orig_file_name", "user_name",
"customer_name", "customer_id", "customer_job_name", "start_time",
"end_time")
    Set conn = CreateObject("ADODB.Connection")
    Set rs = CreateObject("ADODB.Recordset")
    conn.Open DSN, UserName, Password
    OrderByFieldName = DBJobsPreparedFields(OrderByFieldNr)
    'MsgBox (OrderByFieldName)
```

```
    'QueryString = "SELECT * FROM [jobs_prepared] ORDER BY [" &
OrderByFieldNr & "] ASC"

     OrderAscending = True
    Select Case OrderByFieldName
       Case "end_time":   OrderAscending = False
       Case "start_time":   OrderAscending = False
      Case Else: OrderAscending = True
    End Select

    If OrderAscending = True Then
    '    MsgBox ("asc")
        rs.Open "SELECT * FROM [" & DBTable & "] ORDER BY [" &
OrderByFieldName & "] ASC", conn, 1, 3
    Else
     '   MsgBox ("desc")
        rs.Open "SELECT * FROM [" & DBTable & "] ORDER BY [" &
OrderByFieldName & "] DESC", conn, 1, 3
    End If

    'rs.Open(QueryString, conn, 1, 3)
    rc = rs.RecordCount - 1
    Set StartOffset = ThisDisplay.GetPointObjectFromName("~~start_offset~~")

    Dim FirstDisplayedRowNr, DBRecordCount As Integer

    'Memory for the selected row, so that after update could be focused once
    'again the right one
    ActiveRowNr = GetActiveRowNr()
    ActiveRecordNr = GetValue("~~" & ActiveRowNr & "DBRowNr~~")
    FirstDisplayedRowNr = 0
    Min = StartOffset.value
    Max = Min + rowCount
    If Max > rc Then
        Min = rc - rowCount + 1
        'MsgBox ("Last row already reached")
        Call SetValue("~~start_offset~~", Min)
    End If
    If Min <> 0 Then
    For ii = 0 To Min - 1
        rs.MoveNext
        FirstDisplayedRowNr = FirstDisplayedRowNr + 1
    Next
    End If
    DBRecordCount = rs.RecordCount
    If DBRecordCount < rowCount Then
        xmax = DBRecordCount - 1
    Else
        xmax = rowCount - 1
    End If
    For x = 0 To rowCount - 1
        For i = 0 To fieldCount - 1
            Call SetValue("~~" & x & "DBRowNr" & "~~", 0)
        Next i
    Next x


    Call SetValue("~~DBRecordCount~~", DBRecordCount - rowCount)
    For x = 0 To xmax
        For i = 0 To fieldCount - 1
            Set temp = ThisDisplay.GetPointObjectFromName("~~" & x &
"DBField" & i & "~~")
            If rs.fields(DBJobsPreparedFields(i)).value <> "" Then
            'MsgBox (DBJobsPreparedFields(i) + " " +
rs.fields(DBJobsPreparedFields(i)).value)
```

```
                temp.value = CStr(rs.fields(DBJobsPreparedFields(i)).value)
            Else
                temp.value = ""
            End If

          Call SetValue("~~" & x & "DBRowNr" & "~~", FirstDisplayedRowNr + x)
        Next
        rs.MoveNext
    Next
    Set temp = Nothing
    rs.Close
    Set rs = Nothing
    conn.Close
    Set conn = Nothing


    StartOffsetValue = Int(GetValue("~~start_offset~~"))
    RowNrToSet = ActiveRecordNr - StartOffsetValue
    'MsgBox RowNrToSet

    If RowNrToSet < 0 Then RowNrToSet = 0
    If RowNrToSet > rowCount - 1 Then RowNrToSet = rowCount - 1
    Dim StrRadioButton As String
    StrRadioButton = RowNrToSet & "DBRadioButton"
   Set temp4 = ThisDisplay.GetDynamicObjectFromName(StrRadioButton)
   If temp4.objectName = StrRadioButton Then
        If CalledFrom <> "Slider" Then
            'Because when called from slider, comes Total Failure
            Call temp4.SimulateClick(0)
        End If

   End If


End Sub
```

#### 7.3.1.3 Otevření webové stránky pomocí ActiveX prvku Microsoft Web Browser

```
Public Sub ShowFileInWebBrowser()
    Dim FileToLoad, FileToLoadWithPath As String
    Dim fso
    FileToLoad = GetFileToLoad()
    Set fso = CreateObject("Scripting.FileSystemObject")
    FileToLoadWithPath = JOBS_PREPARED_JDF_FILES_PATH + "\" + FileToLoad
    With fso
        If .FileExists(FileToLoadWithPath) Then


    'MsgBox (FileToLoad)
            If FileToLoad <> "" Then
                If MsgBox("You are about to edit a file" + Chr(13) + _
                      "The file name is: " + FileToLoad + Chr(13) + Chr(13) +
_
                      "Do you want to proceed?", 36, "Load file") = vbYes
Then
WebBrowser_URLToLoad = "http://localhost/jdf/gen_edit_file.php" + _
                                            "?Select_File_Name=" + FileToLoad
                    Call GwxDBEditJDFFile_MainForm.ShowURL
                    GwxDBEditJDFFile_MainForm.Show
                End If
            End If
        Else
            MsgBox ("The file does not exist" + Chr(13) + _
                    "File Name: " + FileToLoadWithPath)

        End If
```

```
    End With

End Sub
Public Sub ShowURL()

    Dim tempProgressBarValue As Integer

    'MsgBox (WebBrowser_URLToLoad)
    GwxDBEditJDFFile_MainForm.WebBrowser1.Navigate (WebBrowser_URLToLoad)
    GwxDBEditJDFFile_MainForm.Repaint
    BusyStatus = WebBrowser1.Busy
    If BusyStatus = True Then
        lblStatus.Caption = "Status: Loading the file. Please wait ..."
        pbarLoadingStatus.Visible = True
        While tempProgressBarValue < 100

            pbarLoadingStatus.value = tempProgressBarValue
            tempProgressBarValue = tempProgressBarValue + 1

        Wend

    End If
End Sub

Private Sub WebBrowser1_NavigateComplete2(ByVal pDisp As Object, URL As
Variant)

    BusyStatus = False
    pbarLoadingStatus.value = 0
    pbarLoadingStatus.Visible = False

    lblStatus.Caption = "Status: File loaded."

    WebBrowser1.SetFocus
    TextBox1.value = WebBrowser1.LocationURL



End Sub
```

## *7.4 Přehledové tabulky*

### 7.4.1 Příloha A-1 – Sběrnice - fyzická struktura a časové chování

| *Srovnávací kritéria* | *BITBUS* | *CAN* | *DIN-Messbus* | *LON* | *PROFIBUS-DP* | *PROFIBUS-PA* | *Ethernet Powerlink* |
|---|---|---|---|---|---|---|---|
| *Topologie sítě* | *linie, strom* | *linie* | *linie, strom* | *linie (2-dráty) libovolna* | *linie* | *linie, strom, hvězda* | *Hvězda nebo strom s úrovní dvou Hubů* |
| *max. počet účastníků (M+S)* | *1M + 32 až 240 S + 10 Repeater* | *nespecifikováno* | *32 (až 4096 Slaves při více vrstvách)* | *32 385 Účastníků (127 pro podsíť)* | *124 (4 Segm., 3 Repeater), max. 32 pro Seg.* | *32 pro Segment* | *253* |
| *maximální délka vedeni (bez opakovačů)* | *1,2 km (62,5 kbit/s) 300 m (375 kbit/s)* | *1 km (50 kbit/s) 40 m (1 Mbit/s)* | *2 km (9,6 kbit/s) 500 m (1 Mbit/s)* | *2 km (78 kbit/s) TP 6,1 km (5,48 kbit/s) (Plastový LWL)* | *1,2 km (93,75 kbit/s), 100 m (12 Mbit/s)* | *1,9 km* | *100m jeden segment* |
| *maximální délka vedeni (s opakovači)* | *13 km (62,5 kbit/s) 900 m (375 kbit/s)* | *nespecifikováno* | *libovolná* | *téměř libovolná* | *10 km (93,75 kbit/s)* | *ca. 5,7 km* | *dtto* |
| *max. vzdálenost mezi účastníky* | *1,2 km (62,5 kbit/s) 300 m (375 kbit/s)* | *1 km (50 kbit/s), 40 m (1 Mbit/s)* | *2 km (9,6 kbit/s) 500 m (1 Mbit/s)* | *2 km* | *jako délka vedení* | *jako délka vedení* | *dtto* |
| *médium sběrnice (vedení)* | *2 TP/1 TP + 3 TP (pro Opakovač LWL* | *stíněná kroucená dvojlinka, Powerline, LWL* | *2 x Twisted Pair stíněná, LWL* | *Twisted Pair, rádiové vlny, IR, Powerline, LWL* | *stíněná kroucená dvojlinka, LWL* | *stíněná kroucená dvojlinka* | *twisted pair* |
| *rozhraní* | *RS 485, Modem, LWL* | *ISO 11898,11519-2, ISO/DIS 11992 RS 485* | *RS 485,plně duplexní* | *RS 485* | *RS 485* | *IEC 1158-2* | *Ethernet* |
| *Přenosová rychlost* | *62,5 kbit/s 375 kbit/s (typicky) 1,5 Mbit/s* | *20 kbit/s až 1 Mbit/s* | *9,6 kbit/s (až 1 Mbit/s)* | *600 bit/s až 1,25 Mbit/s* | *9,6 kbit/s až 12 Mbit/s* | *31,25 kbit/s* | *100 Mbit/s* |
| *typický obnovovací čas (např. 8 Účastníků, 4 Byte data)* | *cca. 25 ms (375 kbit/s)* | *cca. 1,3 ms (1 Mbit/s, hochprior)* | *2,6 ms při 500 kbit/s)* | *cca. 70 ms* | *cca. 4,7 ms při 500 kbit/s* | | *400µs* |

### *7.4.2* Příloha A-2 – Sběrnice - vlastnosti protokolu a strategické kriteria

| *Srovnávací kritéria* | *BITBUS* | *CAN* | *DIN-Messbus* | *LON* | *PROFIBUS-DP* | *PROFIBUS-PA* | *Ethernet Powerlink* |
|---|---|---|---|---|---|---|---|
| *Hierarchie* | *centrální (Single-Master)* | *decentrální (Multi-Master)* | *centrální (Single-Master)* | *decentrální (Multi-Master)* | *centrál-ní/decentrální (Single-Master)* | *centrál-ní/decentrální (Single-Master)* | *-* |
| *řízení přístupu* | *SDLC/Polling, deterministický* | *CSMA/CA, zprávově orientova-ný* | *volný Polling, deterministický* | *CSMA/CD (modifikovaný)* | *cyklický Polling, deterministický* | *cyklický Polling, deterministický* | *SCNM* |
| *délka telegramu (užitná data)* | *0 až 248 Byte* | *0 až 8 Byte* | *1 - 128 Byte, auto-mat. dělení bloku* | *1 až 228 Byte typ. 11 Byte* | *0 až 246 Byte* | *0 až 246 Byte* | *max 1500 byte* |
| *bezpečnost* | *HD = 4* | *HD = 6* | *HD = 4* | *HD = 4* | *HD = 4* | *HD = 4* | *-* |
| *mezinárodní norma* | *IEEE 1118* | *ISO 11898* | *Euronorma* | *IEC 62026* | *EN 50170* | *EN 50170 / IEC 1158-2* | *zatím není* |
| *podporováno frimami a zájmovými sdruže-ními* | *BEUG e. V. SPS-Hersteller, PC-Hersteller, Chip-Hersteller* | *CiA, SAE Autoindustrie, OSEK, LAV, přes 1000 výrobců* | *ADM e. V. PTB Berlin, MFP Wunstorf* | *Echelon, USA Gesytec, D, LNO, LUI* | *PNO, Siemens AG* | *PNO, Siemens AG* | *B&R* |
| *rozšíření* | *mezinárodní* | *mezinárodní* | *Německo* | *USA, globálně* | *Německo, globálně* | *Německo, globálně* | *globálně* |
| *Cena [1] za připojené zařízení* | *cca 100 až 500 EUR* | *nízká (cca 10 EUR)* | *cca 20 až 600 EUR* | *od cca 15 EUR* | *70 až 700 EUR* | *140 až 450 EUR* | *-* |
| *zvláštní poznámky* | *jednoduchá, cenově výhodná, spolehlivá* | *vysoce akceptovaná a široce rozšířená, mnoho CAN-Controllerů* | *jednoduchá* | *vzdálené napájení, možnost použití uživatelského soft-ware u všech uzlů.* | *přenosová rychlost až 12 MBit/s* | *vlastní bezpečnost* | *-* |

[1] orientační cena v roce 2000