TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

PRÁCE S DATOVÝMI OBJEKTY V PLC

Bakalářská práce

Studijní program: B2646 – Informační technologie
Studijní obor: 1802R007 – Informační technologie

Autor práce: **Stanislav Mareš**
Vedoucí práce: Ing. Leoš Beran, Ph.D.

Liberec 2015

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

DATA OBJECTS MANAGEMENT IN PLC

Bachelor thesis

Study programme: B2646 – Information Technology
Study branch: 1802R007 – Information Technology

Author: **Stanislav Mareš**
Supervisor: Ing. Leoš Beran, Ph.D.

Liberec 2015

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2014/2015**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Stanislav Mareš**
Osobní číslo:	**M12000161**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Práce s datovými objekty v PLC**
Zadávající katedra:	**Ústav mechatroniky a technické informatiky**

Zásady pro vypracování:

1. Vytvořte programové vybavení pomocí jazyka ST pro správu datových objektů v PLC.
2. Jasně definujte příkazy, parametry a stavy programu.
3. Program musí umožnit: vytvoření, mazání, zvětšení, zmenšení datového objektu dle potřeb uživatele; zápis, čtení a mazání dat libovolného formátu v datovém objektu;
4. Výsledky práce musí být pečlivě zdokumentovány včetně nápovědy pro případného uživatele.

Rozsah grafických prací: **dle potřeby dokumentace**

Rozsah pracovní zprávy: **30–40 stran**

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] **AUTOMATION, B&R. Controls - training text. Austria: [s.n.], 2008. 205 s.**

[2] **BERNECKER + RAINER INDUSTRIE-ELEKTRONIK GES.M. B. H. B&R Help: DataObj. 2013. vyd. 2013.**

[3] **ŠMEJKAL, Ladislav a Marie MARTINÁSKOVÁ. PLC a automatizace. 1. díl. Praha: BEN, 1999. ISBN 80-86056-58-9.**

[4] **JOHN, Kharl-Heinz; TIEGELKAMP, Michael. IEC 61131-3 Programming Industrial Automation Systems : Concepts and Programming Languages, Requirements for Programming Systems, Decision - Making Aids. 2nd Edition. NewYork : Springer, 2010. 390 s. ISBN 978-3-642-12015-2.**

Vedoucí bakalářské práce: **Ing. Leoš Beran, Ph.D.**
Ústav mechatroniky a technické informatiky

Konzultant bakalářské práce: **Ing. Martin Diblík, Ph.D.**
Ústav mechatroniky a technické informatiky

Datum zadání bakalářské práce: **10. října 2014**

Termín odevzdání bakalářské práce: **15. května 2015**

prof. Ing. Václav Kopecký, CSc.
děkan

L.S.

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 10. října 2014

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum:

Podpis:

Poděkování

Chtěl bych poděkovat vedoucímu této bakalářské práce Ing. Leoši Beranovi, Ph.D. a konzultantovi Ing. Martinu Diblíkovi, Ph.D. za odborné vedení, přínosné rady, ochotu a vstřícnost při realizaci touho díla.

Abstrakt

Cílem této bakalářské práce je vytvořit software, který bude sloužit jako správce datových objektů v programovatelných logických automatech neboli PLC.

Daný problém jsem rozložil do dvou menších podprogramů, které spolu komunikují. První podprogram poskytuje rozhraní pro uživatele a druhý zajišťuje práci přímo s datovými objekty.

Výsledkem je program, který umožňuje základní operace s datovými objekty jako je jejich vytváření a mazání nebo čtení a zápis datových záznamů v datových objektech.

Z důvodu využití statických datových struktur je program poměrně rychlý, ale neefektivně využívá paměť.

Hlavním přínosem je výrazné zjednodušení a zvýšení efektivity práce s datovými objekty.

Klíčová slova

programovatelný logický automat, datový objekt, Automation Studio, strukturovaný text, statická datová struktura

Abstract

The goal of this bachelor thesis is to develop a software that will serve as a data object manager in programmable logic controllers also known as PLCs.

I have divided given problem into two smaller subprograms which communicate with each other. The first subprogram provides a user interface and the second subprogram provides work directly with data objects.

The result is a program that allows basic data object manipulation, such as creation and deletion or reading and writing of data records in data objects.

Thanks to the usage of static data structures the program is relatively fast but it is also inefficiently using memory.

The main benefit is significant simplification and increase of work effectivity with data objects.

Key words

Programmable logic controller, data object, Automation Studio, structured text, static data structure

Obsah

Seznam obrázků

Seznam zkratek

B&R – Bernecker a Rainer

PLC – Programmable Logic Controller

ST – Structured Text

AB – Automation Basic

DRAM - Dynamic Random Access Memory

USRRAM – User Random Access Memory

Úvod

Tématem této bakalářské práce je správa datových objektů v programovatelných logických automatech. Tyto relativně malé počítače se pro své široké využití hojně používají po celém světě. Uplatňují se zejména v průmyslových podnicích, kde mohou řídit výrobní linky či různé stroje. Dále jsou schopné ovládat například výtahy a další podobná zařízení. V dnešní moderní době je taktéž můžeme najít u inteligentních domů, kde mají na starost zabezpečení, regulaci vytápění nebo řízení osvětlení. Své využití nacházejí zkrátka všude, kde je potřeba cokoli automatizovat. Datové objekty jsou pak konkrétním softwarovým vybavením, které slouží ke zpracovávání a uchovávání dat získaných například z různých měření apod.

Cílem práce je vytvoření programového vybavení pomocí jazyka Strukturovaný Text (ST), které slouží pro správu datových objektů v PLC.

Celý program jsem rozdělil do dvou menších podprogramů. První z nich poskytuje rozhraní pro uživatele, přes které lze program ovládat a kontrolovat jeho vstupy a výstupy. Druhý podprogram potom zajišťuje práci přímo s datovými objekty a udržuje si informace o doposud vytvořených objektech a datech v nich uložených.

Programovatelné logické automaty a programování samotné jsou obory, které mě zajímají a baví. Myslím, že je to rychle rozvíjející se směr, který má v dnešním světě budoucnost. To jsou hlavní důvody výběru tohoto tématu pro moji bakalářskou práci.

1 Použité technologie

1.1 Automation Studio

Automation studio je software od společnosti B&R sloužící pro vývoj programů, které jsou určeny programovatelným logickým automatům. Jde o velmi rozsáhlý program s mnoha funkcemi, i přesto je však, podle mého názoru, uživatelsky velmi přívětivý. Nabízí značnou podporu programovacích jazyků, ve kterých lze programy psát, například již zmiňovaný Strukturovaný Text, B&R Automation Basic (AB), ANSI C a další.

1.2 Strukturovaný Text

Tento jazyk je jedním z pěti jazyků normy IEC 61131-3 navržených pro programovatelné logické automaty. Všechny jazyky této „rodiny“ sdílejí společné prvky. Strukturovaný Text se řadí mezi jazyky vyšší úrovně a syntakticky se podobá jazykům C a Pascal. Podporuje vše, co je potřeba k základnímu programování, jako například iterační cykly, podmínky, funkce apod.

1.3 Datové objekty

Datové objekty jsou svojí strukturou velmi podobné binárním souborům. Představují jeden ze dvou způsobů dynamického ukládání dat v Automation Studiu. Další alternativou takové práce s daty jsou soubory. Každý způsob má své výhody i nevýhody a je na programátorovi, zda zvolí způsob první, druhý, nebo jejich kombinaci. Datové objekty mají oproti souborům značnou výhodu z hlediska bezpečnosti dat. PLC si totiž automaticky vytváří zálohu těchto datových struktur. Díky tomu mohou být data zpětně obnovena například po výpadku proudu během práce. U souborů by v takové situaci došlo ke ztrátě dat.

1.4 Knihovna DataObj

Základní práci s datovými objekty zajišťuje vestavěná knihovna s názvem *DataObj*. Umožňuje programátorovi datové objekty například vytvářet, mazat nebo do nich zapisovat data a zpětně je číst pomocí tzv. funkčních bloků. Příklady těchto bloků mohou být: *DatObjCreate*, *DatObjDelete, DatObjWrite* nebo DatObj*Read*. Všechny funkční bloky pracují asynchronně - to znamená, že ne v každém cyklu jsou připravené splnit požadovanou operaci. Toto by měl mít programátor na paměti. V případě, že je právě

funkční blok zaneprázdněn, je třeba zajistit, aby se v příštích cyklech spouštěl znovu, dokud požadovanou operaci neprovede.

2 Současná práce s datovými objekty

Jak již jsem zmiňoval v předchozí kapitole, knihovna *DataObj* nám poskytuje několik funkčních bloků pro základní práci s datovými objekty. Jedná se například o vytváření nebo mazání objektů. Pro menší programy je taková funkcionalita možná dostatečná, nicméně pro ty rozsáhlejší již začíná mít silné nedostatky.

Největší problém při práci s touto knihovnou spočívá v tom, že si nikde neukládá informace o datových objektech a datových záznamech v nich uložených. Mezi takové informace patří například název, délka či místo uložení datového záznamu v objektu. Zde bych znovu připomněl, že datové objekty jsou soubory binárními. Pro programátora to znamená, že musí znát přesně strukturu dat v datovém objektu, aby s nimi mohl správně pracovat. V případě neznalosti rozložení dat by se mohlo stát, že bude programátor načítat jiná data, nebo si dokonce nějaká data přepíše.

I přesto, že si funkční bloky z knihovny *DataObj* nikde neukládají záznamy o názvu, délce či struktuře uložených dat, vyžadují tyto informace k tomu, aby mohly správně fungovat. Je tedy na programátorovi, aby si vše potřebné zálohoval. U menších programů, kde by datových objektů nebylo mnoho, by se nejednalo o tak velký problém. Ale představme si program s 10000 objekty a v každém objektu 500 datových záznamů. V takovém rozsahu by to byl již problém značný.

Další věcí je ošetřování chybových stavů. Nastane-li nějaká chyba, funkční blok pouze zahlásí komplikaci a odkáže na číslo chyby. Pak už záleží na programátorovi, jak daný problém vyřeší.

Celkově vzato, má knihovna spoustu nedostatků, které za ní musí ošetřovat programátor. A řešit vždy znovu ty samé problémy, když se rozhodneme knihovnu použít, je velmi zdlouhavé a neefektivní. A jsme u důvodu vzniku této práce.

2.1 Manažer datových objektů

Odpovědí na výše zmíněné záležitosti je Manažer datových objektů, který řeší problémy za uživatele. Ten již nepracuje přímo s funkčními knihovny *DatObj*, ale vše je plně automatické. Uživatel má k dispozici pouze jednoduché rozhraní, pomocí něhož může program ovládat.

3 Tvorba programu

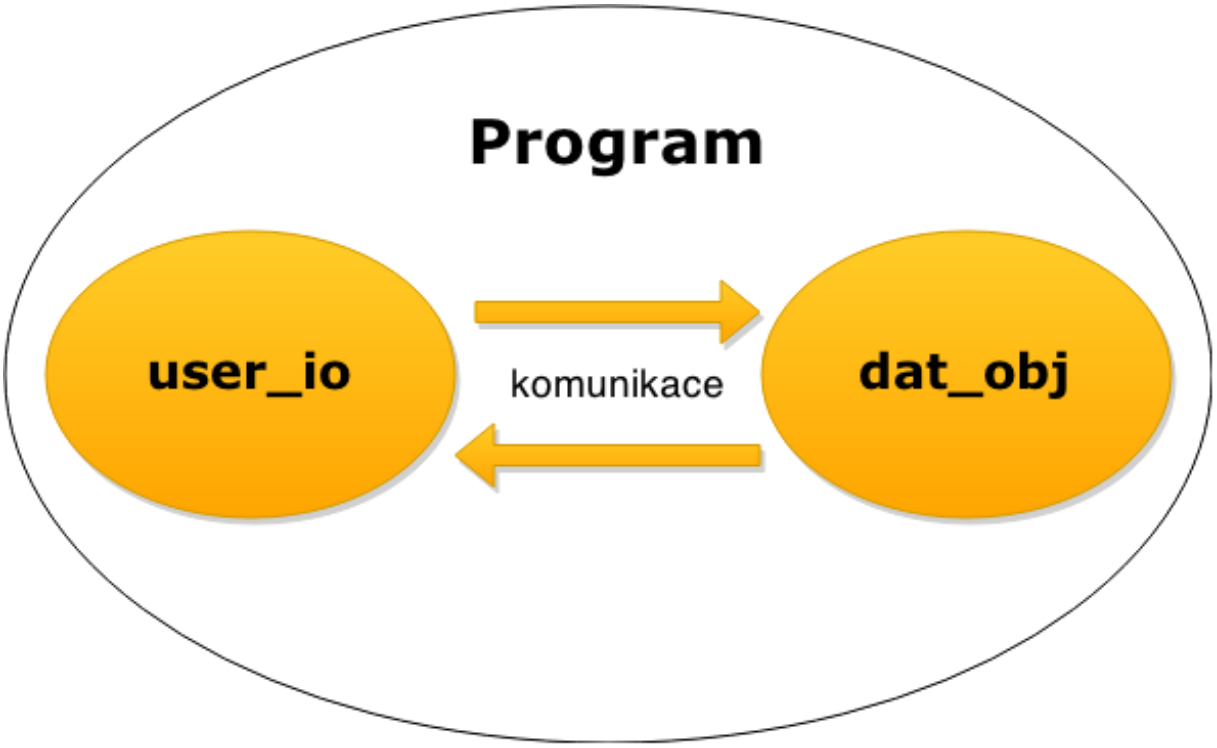
3.1 Programy v Automation Studiu

Všechny programy v tomto vývojovém prostředí se mohou skládat až ze tří částí. První se nazývá *INIT* a slouží k inicializaci proměnných po restartu PLC. Druhá část - *CYCLIC* je ze všech tří tou nejdůležitější, jelikož se zde nachází hlavní kód, který je cyklicky vykonáván a tím dává programovatelnému automatu určitou funkcionalitu. Obě zmíněné části jsou povinné. Třetí část, která je nepovinná, se nazývá *EXIT* a vykonává se před vypnutím PLC. Do této části se může například umístit uvolnění dynamicky alokované paměti atd.

Běžící programy se v PLC nazývají úlohy. Každý program se řadí do tzv. cyklické třídy, kterých je celkem osm. Každá z nich má přiřazený jistý čas, který určuje, dokdy má být daná úloha dokončena a s jakou frekvencí se má její kód spouštět. Čím kratší čas, tím má úloha vyšší prioritu. To znamená, že je procesorem obsloužena dříve než úlohy s prioritou nižší. Samozřejmě je na programátorovi, aby dobře promyslel, do jakých cyklických tříd své programy zařadí. Musí brát také ohled na náročnost svého programu. Mohlo by se totiž stát, že by daná úloha nemusela být stihnuta z důvodu vysoké výpočetní náročnosti a krátké doby dokončení. To by poté vedlo k neúplným výsledkům a ke špatnému chodu celého programu.

3.2 Návrh programu

Celý program jsem rozdělil do dvou logických celků - podprogramů, z nichž každý se stará o určitou část problému. Oba podprogramy spolu komunikují prostřednictvím zasílání zpráv. První z nich se jmenuje *user_io*. Tento podprogram se stará o interakci s uživatelem a kontroluje jeho vstupy a výstupy. Dále poskytuje uživateli jednoduché rozhraní, pomocí něhož může program ovládat a reagovat na vzniklé problémy. Druhý podprogram, nesoucí název *dat_obj,* zajišťuje přímou práci s datovými objekty a také si ukládá všechny potřebné informace o doposud vytvořených objektech a datech v nich uložených.



Obrázek 1: Logické rozdělení programu

3.3 Podprogram user_io

Tento podprogram je zařazen do cyklické třídy č. 4 a to znamená, že se bude vykonávat s frekvencí 1 s. Program se skládá celkem ze šesti stavů. Pro doplnění zde uvedu, že stavem programu se myslí určitá jeho fáze, v které setrvává, dokud vyhodnocovací logika stav nezmění. Tato logika může stav programu změnit například zásahem uživatele nebo v důsledku nových dat, které byly právě vypočteny. Názvy stavů jsou: *DISABLE, ENABLE, CHECK, FIND, FINISH* a *ERROR*.

3.3.1 DISABLE

Do tohoto stavu program přechází vždy, když uživatel úspěšně odešle data ke zpracování a setrvává zde, než program *dat_obj* dokončí svoji činnost, nebo dokud nevrátí chybové hlášení. Pokud taková situace nastane, program přejde do stavu *ERROR*. V případě, že zpracování proběhne úspěšně, uživateli jsou poskytnuta požadovaná data a zobrazí se mu zpráva o úspěchu, případně ještě nějaká doplňující informace. Dále jsou všechny jeho příkazy a editační pole nastaveny na výchozí hodnotu. Poté program přechází do stavu *ENABLE*.

3.3.2 ENABLE

V tomto stavu program čeká, než uživatel zadá nějaký příkaz. Tím může být například vytvoření datového objektu či jeho smazání. Poté je uživateli zobrazena zpráva o zpracovávání dat a program přechází do stavu *CHECK*.

3.3.3 CHECK

Zde se nachází logika, která zastupuje první fázi kontroly a ověřuje, zda uživatel zadal všechna potřebná data na základě zvoleného příkazu. Například při výběru příkazu „vytvoř datový objekt“, musí uživatel poskytnout programu název a délku tohoto nového objektu. V případě, že nějaké z polí nechá prázdné, program přejde do stavu *ERROR*. Jestliže je vše vyplněno v pořádku, přejde program do stavu *FIND.*

3.3.4 FIND

V tomto stavu probíhá druhá fáze kontroly uživatelových vstupů. Také v této fázi záleží na tom, jakou operaci uživatel požaduje. Opět uvedu příklad na vytváření nového objektu. Nejprve se testuje, zda uživatel nepřekročil stanovený limit datových objektů. Pokud ano, je mu zamezeno vytvořit nový objekt a přejde se do stavu *ERROR*. Jestliže limit překročen nebyl, program kontroluje, zda jméno objektu, které uživatel zvolil, již v programu neexistuje. Je-li vše v pořádku, program pokračuje do poslední fáze, kterou je stav *FINISH.* V opačném případě se opět přechází do stavu *ERROR*.

3.3.5 FINISH

FINISH je závěrečným stavem, ve kterém se zašlou data spolu se zprávou o dokončení programu *dat_obj*. Jakmile se tak stane, program *dat_obj* začne vykonávat svojí činnost a zašle zpět zprávu programu *user_io* o této události. Po obdržení zprávy program *user_io* přechází do stavu *DISABLE* a tím je zakončen jeden průchod tohoto programu. Celý proces se poté může opakovat.

3.3.6 ERROR

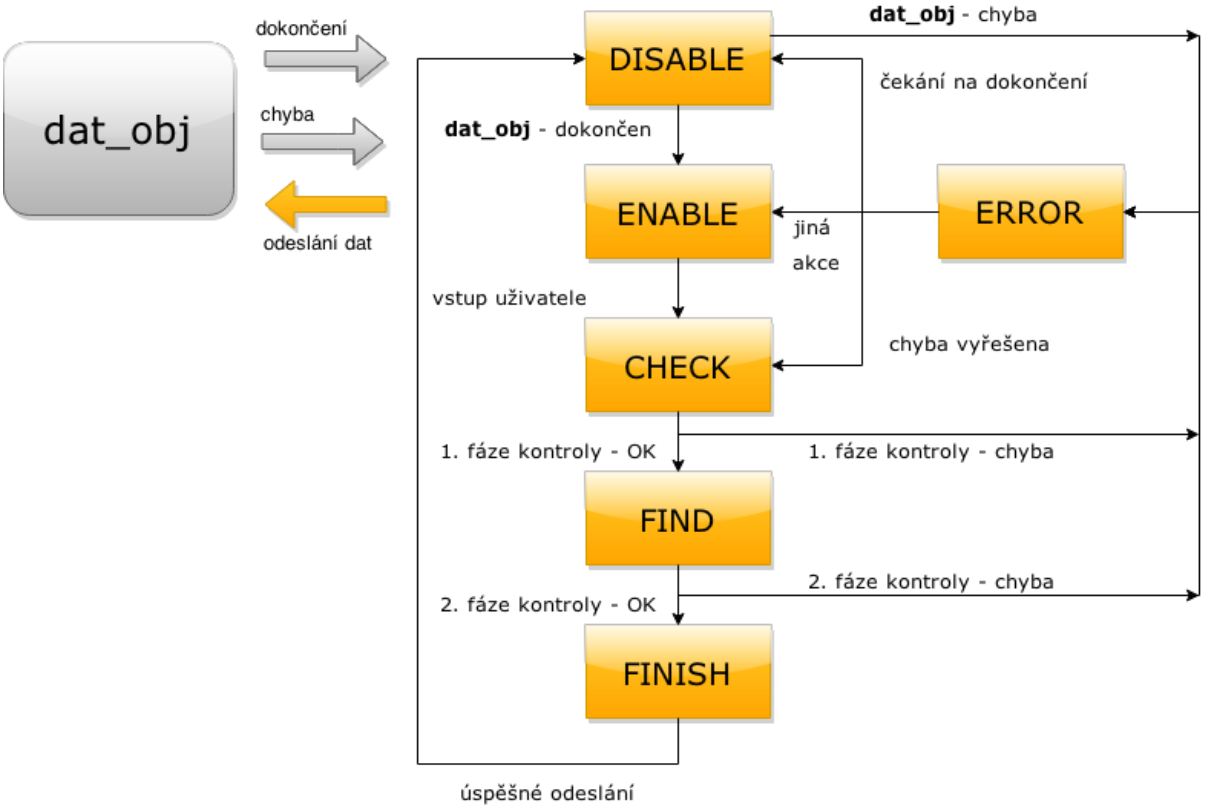
Ve stavu *ERROR* se nachází logika, která má za úkol informovat uživatele o typu vzniklé chyby a poskytnout mu nápovědu pro její vyřešení. V závislosti na typu chyby program buď v tomto stavu setrvává, a to dokud uživatel na chybu nezareaguje, nebo se pouze zobrazí zpráva o vzniklé chybě a program přechází do stavu *ENABLE.*

Prvním typem chyby je taková, která mohla vzniknout uživatelovou nepozorností. Například, že zapomněl vyplnit nějaký potřebný údaj. V takovém případě

se čeká na uživatelovo potvrzení, že daný problém odstralil. Poté program přechází do stavu *CHECK,* aby znovu překontroloval uživatelovy vstupy. Odtud buď pogram pokračuje do další fáze kontroly nebo opět přejde do stavu *ERROR* v případě, že se uživateli nepodařilo chybu odstranit či nastala ještě nějaká další.

K další chybě může dojít také při zpracovávání dat v programu *dat_obj*. Například když v daném objektu není dostatek místa na uložení dalšího datového záznamu. V tomto případě je uživateli nabídnuta možnost volby, zda chce daný objekt zvětšit či nikoliv. Po výběru jedné z možností program *user_io* přechází do stavu *DISABLE* a čeká se na dokončení programu *dat_obj*.

Třetí typ chyby vzniká, dosáhne-li uživatel některého z limitů programu. Například limitu na vytvoření datových objektů nebo datových záznamů. V takovém případě se uživateli zobrazí chybová zpráva a program přechází rovnou do stavu *ENABLE*. To je z toho důvodu, že uživatel musí na tento typ chyby reagovat nějakou jinou operací, například smazáním datového objektu s cílem uvolnit místo.



Obrázek 2: Stavový diagram programu user_io

3.4 Podprogram dat_obj

Tento podprogram je také umístěn do cyklické třídy č. 4 a skládá se celkem z deseti stavů: *WAIT*, *INFO*, *COMMAND*, *CREATE*, *READ*, *WRITE*, *OFFSET*, *DELETE*, *UPDATE* a *RESIZE*.

3.4.1 WAIT

V tomto stavu program čeká, než je mu zaslána zpráva o dokončení a odeslání dat z programu *user_io.* Když se tak stane, program nejprve vyhodnotí uživatelův příkaz. Jestliže uživatel zvolil operaci vytvoření nového datového objektu, přechází program rovnou do stavu *CREATE.* Pokud došlo k vybrání kterého koliv jiného příkazu, pokračuje se do stavu *INFO.*

3.4.2 INFO

Zde program získává informace o datových objektech za pomoci funkčního bloku *DatObjInfo* z knihovny *DataObj*. Takovými informacemi je například identifikátor datového objektu nebo délka objektu. Tyto informace jsou potřebné pro další fázi programu. Funčknímu bloku je nastaven jediný parametr, kterým je název datového objektu. Jakmile vrátí funkční blok status roven nule, znamená to, že úspěšně dokončil svoji činnost a program pokračuje do stavu *COMMAND.*

3.4.3 COMMAND

Tento stav obsahuje pouze vyhodnocovací logiku, která určí, do jakého stavu má program na základě uživatelova příkazu přejít. Jestliže uživatel zvolil operaci čtení, zápis či vymazání datového záznamu, pokračuje program do stavu *OFFSET*. Jinak přechází do stavu *DELETE.*

3.4.4 CREATE

Program zde za pomoci funkčního bloku *DatObjCreate* z knihovny *DataObj* vytváří nové datové objekty. Nejprve se nastavují příslušné parametry funkčnímu bloku. Nejdůležitější z nich - jméno a délka, jsou poskytnuty uživatelem. Následně se funkční blok zavolá a čeká se na jeho dokončení. Jakmile se tak stane, přechází program do stavu *UPDATE*.

3.4.5 READ

Zde se pracuje s funkčním blokem *DatObjRead* z knihovny *DataObj*. Opět se nejprve nastaví příslušné parametry. Nejdůležitější z nich jsou: identifikace, délka čtených dat, datový posun a adresa struktury. Datový posun určí, odkud se mají data začít číst a adresa struktury říká, kam chce uživatel data načíst. Následně je funční blok volán a čeká se na jeho dokončení. Jakmile se tak stane, zašle se zpráva programu *user_io* o dokončení operace a program přechází do stavu *WAIT*. Protože se jedná o operaci čtení, nemusí se nic aktualizovat. Proto je stav *UPDATE* po tomto příkazu vynechán.

3.4.6 WRITE

V tomto stavu je umístěn funkční blok *DatObjWrite* z knihovny *DataObj.* Opět se nejprve nastavují parametry. Ty jsou úplně stejné jako parametry funkčního bloku *DatObjRead.* Rozdíl je pouze v tom, že uživatel poskytuje adresu struktury, ze které chce data zapsat do datového objektu. Poté se daný funkční blok zavolá, a jakmile dokončí svojí práci, program přechází do stavu *UPDATE*.

3.4.7 OFFSET

Tento stav poskytuje informaci o tom, kde přesně jsou datové záznamy v datových objektech umístěny. Tato informace je důležitá zejména při čtení a zápisu dat z/do datových objektů. Vše se vypočítává za pomoci speciálního algoritmu, který je podrobně popsán v kapitole 0. Na základě toho, jaký příkaz uživatel zvolil, přechází poté program do stavu *READ*, *WRITE* nebo *UPDATE.* V případě zjištění nedostatečné kapacity datového objektu programu nejprve přejde do stavu *RESIZE.*

3.4.8 RESIZE

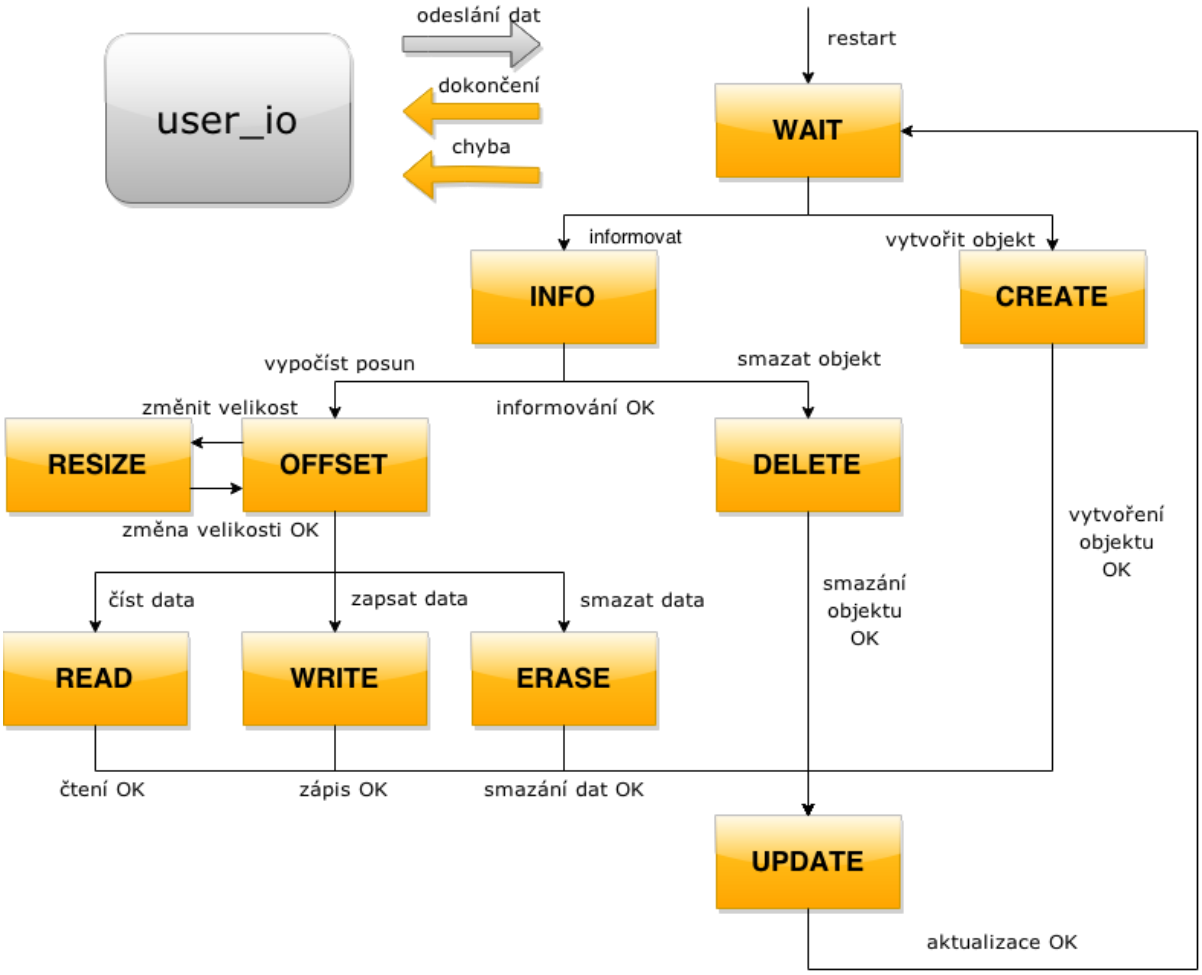
Tento stav zajišťuje zvětšení nebo zmenšení datového objektu na základě potřeb uživatele. Poté se přechází zpět do stavu *OFFSET*. Detailní popis o algoritmu zajišťující změnu velikosti naleznete v kapitole 0.

3.4.9 DELETE

Zde se pracuje s funkčním blokem *DatObjDelete* z knihovny *DataObj*. Nastaví se jediný parametr a tím je identifikátor objektu, který chce uživatel smazat. Tento identifikátor se získá ze stavu *INFO* na základě jména objektu, které poskytne uživatel. Dále se funkční blok zavolá, čeká se na jeho dokončení. Poté program přechází do stavu *UPDATE*.

3.4.10 UPDATE

V tomto stavu se aktualizuje seznam, který obsahuje nezbytné informace o objektech a datových záznamech v nich uložených. Tyto informace jsou důležité pro fungování celého programu. Na základě toho, jaký uživatel zvolil příkaz, se seznam příslušným způsobem aktualizuje. Detailní popis naleznete v kapitole 4.1. Nakonec se zašle zpráva programu *user_io* o dokončení operace a program přejde do stavu *WAIT*. Tím je dokončen celý jeden průchod programem *dat_obj*.



Obrázek 3: Stavový diagram programu dat_obj

4 Implementace

V této kapitole detailněji popíšu implementaci některých významných algoritmů, které tvoří důležité části programu. Mezi takové části patří: udržovaní seznamu všech datových objektů a datových záznamů, způsob výpočtu volného místa v datových oblastech objektů a změna velikosti datových objektů na základě potřeby uživatele.

4.1 Seznam datových objektů a datových záznamů

Udržování seznamu informací o datových objektech a datových záznamech je nezbytným předpokladem pro správné fungování programu. Na jeho informacích jsou totiž závislé všechny použité funkční bloky z knihovny *DataObj*, které jsou zodpovědné za manipulaci s datovými objekty a jejich daty. Tato datová struktura - je polem fixní délky 100, kde každý prvek je složen ze tří dílčích podstruktur.

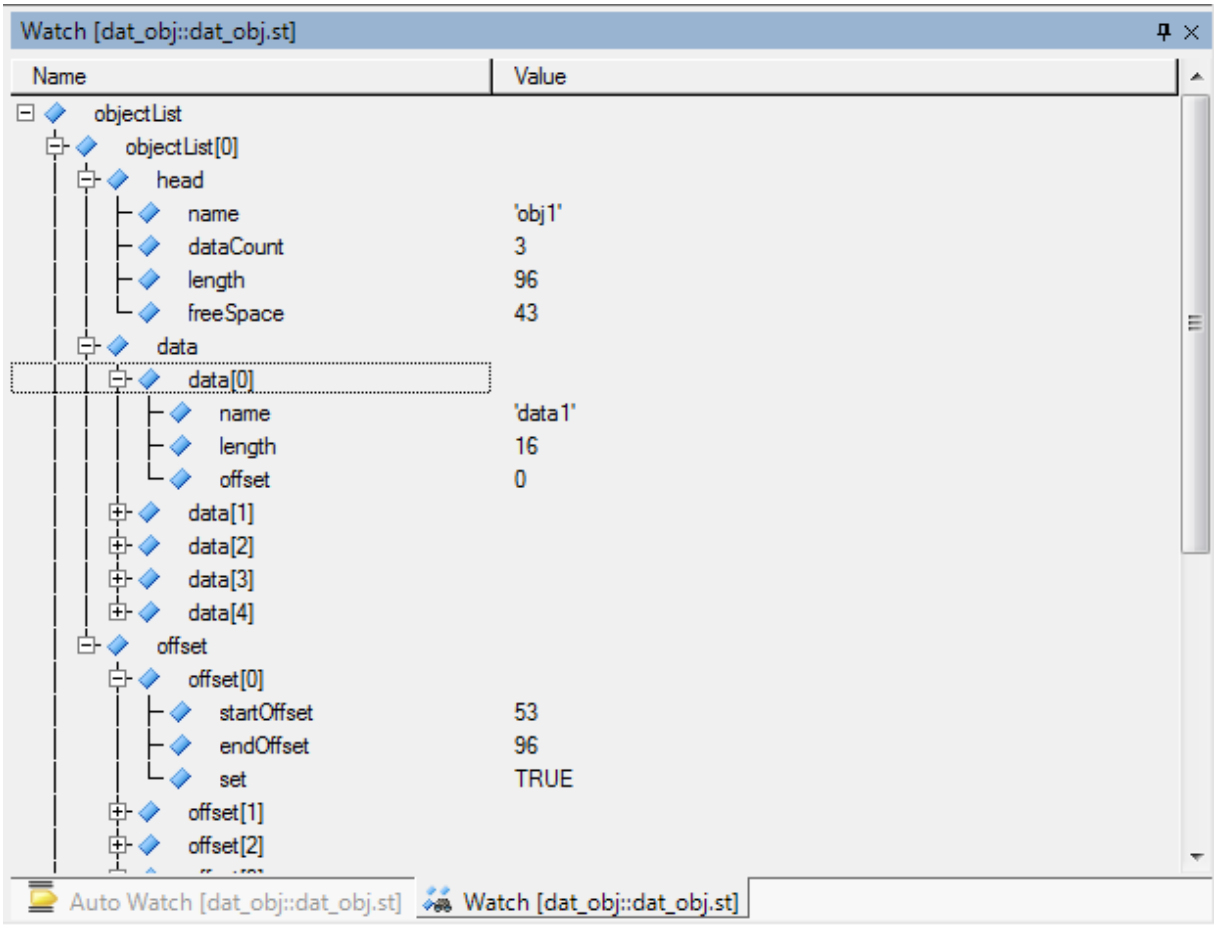
První z nich je hlavička objektu a skládá se ze čtyř proměnných již jednoduchých datových typů. Jsou jimi: jméno objektu typu *STRING[10]*, počet aktuálně uložených datových záznamů v objektu typu *UDINT,* celková délka objektu typu *UDINT* a aktuální volné místo v objektu také typu *UDINT*.

Druhá datová podstruktura je tvořena polem fixní délky 50, které slouží k ukládání informací o datových záznamech vytvořených v datovém objektu. Každý prvek tohoto pole se skládá ze tří proměnných jednoduchých datových typů. Patří sem: název datového záznamu typu *STRING[10]*, délka záznamu typu *UDINT* a posun v datové oblasti objektu typu *UDINT*.

Třetí datovou podstrukturu tvoří rovněž pole fixní délky 50. Zde se ukládají jednotlivé volné intervaly v datové oblasti objektu, kam je možno zapisovat data. Každý prvek pole je složen ze tří proměnných jednoduchých datových typů. První proměnná je typu BOOL a říkám nám, zda je prvek pole obsazen či nikoliv. Druhá je typu *UDINT* a udává, kde přesně začíná interval volného místa. Třetí proměnná je typu *UDINT* a ta naopak určuje, kde interval volného místa končí.

Celá datová struktura je kvůli rychlosti uložena za běhu v dynamické paměti neboli DRAM. Při nečekané události, například při výpadku elektrického proudu, je však přesunuta do paměti zálohované baterií tzv. USRRAM. To z důvodu, aby se data neztratila. Dynamická paměť totiž bez dodávání energie svůj obsah ztrácí. V případě datového manažeru je více než nevhodné, aby taková situace nastala. Uživatel by totiž

přišel o všechny uložené informace o objektech a datových záznamech. Data by sice v paměti fyzicky zůstala, ale uživatel by se k nim nemohl dostat.



Obrázek 4: Ukázka datové struktury objectList v Automation Studiu

4.2 Výpočet volného místa v datové oblasti

Jak jsem již zmínil v kaptiole 4.1, pro každý objekt jsou ukládány tři skupiny informací. První skupinou jsou informace o samotném objektu. Druhou skupinu představují informace o datových záznamech objektu a tou třetí jsou pak informace o volných intervalech v datové oblasti objektu. Zůstává ale otázkou, jak tyto intervaly vypočítávat a aktualizovat, neboť struktura datové oblasti se neustále mění v důsledku zápisu nových datových záznamů nebo jejich mazáním. K tomu nám slouží speciální algoritmus.

4.2.1 Nový objekt

Když si uživatel vytvoří nový datový objekt, do pole volných intervalů je vždy zapsána hodnota 0 pro začátek intervalu a velikost datového objektu pro jeho konec. To znamená, že je volná celá datová oblast objektu.

4.2.2 Nový datový záznam

Jestliže uživatel zadá operaci vytvoření nového datového záznamu, program nejprve testuje, zda je možné záznam, vzhledem k jeho velikosti, uložit. Pokud ano, program začne postupně procházet pole obsahující volné intervaly a hledá první takový, do kterého je možné uložit datový záznam. Jakmile takový interval nalezne, cyklus se okamžitě ukončí, aby se neplýtvalo procesorem. Poté se ještě musí tento interval aktualizovat takovým způsobem, že se jeho začátek posune o velikost délky nového datového záznamu. Interval se tedy zmenší. Pokud se po posunutí začátek rovná jeho konci, znamená to, že nový záznam zabere celý tento datový prostor, a proto se příslušný interval z pole smaže. To je provedeno tak, že se prvek na tomto indexu pouze nastaví jako neobsazený. Na závěr je funkčnímu bloku *DatObjWrite* poskytnuta informace o tom, kam se má datový záznam uložit.

```
FOR i := 0 TO gDatListSize DO
    IF objectList[objIndex].offset[i].set THEN
        IF objectList[objIndex].offset[i].endOffset
            - objectList[objIndex].offset[i].startOffset >= gUser.userData.dataLength THEN

            dataOffset := objectList[objIndex].offset[i].startOffset;

            objectList[objIndex].offset[i].startOffset := objectList[objIndex].offset[i].startOffset
            + gUser.userData.dataLength;

            IF  objectList[objIndex].offset[i].startOffset = objectList[objIndex].offset[i].endOffset THEN
                objectList[objIndex].offset[i].set := FALSE;
                objectList[objIndex].offset[i].startOffset := 0;
                objectList[objIndex].offset[i].endOffset := 0;
            END_IF;

            EXIT;
        END_IF;
    END_IF;
END_FOR;
```

Obrázek 5: Zdrojový kód algoritmu pro aktualizaci volných intervalů po zápisu datového záznamu

4.2.3 Smazání existujícího záznamu

Jestliže se uživatel rozhodne nějaký datový záznam smazat, uvolní se místo v jeho datové oblasti a program musí aktualizovat pole volných intervalů daného objektu. Tato aktualizace probíhá tak, že program postupně prochází toto pole a kontroluje, zda se neshoduje nějaká hranice intervalu prvku s hranicí datového záznamu, který má být smazán. Pokud se shoduje začátek intervalu prvku pole s koncem intervalu datového záznamu, upraví se tomuto prvku začátek jeho intervalu. Je-li nalezena shoda opačná, tzn. konec intervalu prvku pole je stejný jako začátek intervalu datového záznamu, bude prvku pole upraven jeho konec. Když se taková změna v poli provede, s daným prvkem pole je nutné ještě pokračovat dál a otestotvat, jestli se nové hranice prvku pole neshodují s hranicemi nějakého dalšího prvku. Pokud ano, provede se jeho aktualizace znovu výše uvedeným stejným způsobem. Shoda hranic intervalů může nastat v cyklu vždy maximálně dvakrát. Taková je vlastnost tohoto algoritmu.

Aktualizace hodnot prvků pole jsou velmi důležité, neboť zajišťují slučování intervalů volných oblastí, které se navzájem dotýkají. Může se také samozřejmě stát, že shoda není nalezena ani jednou. V takovém případě se nový volný interval uloží na první neobsazený prvek v poli.

Celý algoritmus pracuje obecně. To znamená, že nezáleží na délce datových záznamů, které si uživatel do objektu ukládá.

```
(* start *)
startOffset := objectList[objIndex].data[dataIndex].offset;
endOffset := objectList[objIndex].data[dataIndex].offset + objectList[objIndex].data[dataIndex].length;

FOR i := 0 TO gDatListSize DO
    IF objectList[objIndex].offset[i].set THEN

        IF startOffset = objectList[objIndex].offset[i].endOffset THEN
            IF changeIndex = -1 THEN
                changeIndex := UDINT_TO_SINT(i);
                objectList[objIndex].offset[i].endOffset := endOffset;
                startOffset := objectList[objIndex].offset[i].startOffset;
            ELSE
                objectList[objIndex].offset[changeIndex].startOffset := objectList[objIndex].offset[i].startOffset;
                objectList[objIndex].offset[i].set := FALSE;
                objectList[objIndex].offset[i].startOffset := 0;
                objectList[objIndex].offset[i].endOffset := 0;
            END_IF;
        END_IF;

        IF endOffset = objectList[objIndex].offset[i].startOffset THEN
            IF changeIndex = -1 THEN
                changeIndex := UDINT_TO_SINT(i);
                objectList[objIndex].offset[i].startOffset := startOffset;
                endOffset := objectList[objIndex].offset[i].endOffset;
            ELSE
                objectList[objIndex].offset[changeIndex].endOffset := objectList[objIndex].offset[i].endOffset;
                objectList[objIndex].offset[i].set := FALSE;
                objectList[objIndex].offset[i].startOffset := 0;
                objectList[objIndex].offset[i].endOffset := 0;
            END_IF;
        END_IF;
    ELSE
        IF firstIndex = -1 THEN
            firstIndex := UDINT_TO_SINT(i);
        END_IF;
    END_IF;
END_FOR;

IF changeIndex = -1 THEN
    objectList[objIndex].offset[firstIndex].startOffset := startOffset;
    objectList[objIndex].offset[firstIndex].endOffset := endOffset;
    objectList[objIndex].offset[firstIndex].set := TRUE;
END_IF;
```

Obrázek 6: Zdrojový kód algoritmu pro aktualizaci volných intervalů po smazání záznamu

4.2.4 Změna velikosti datového objektu

Během toho, jak uživatel zapisuje a maže data v datových objektech, se může stát, že u některého z objektů vyčerpá jeho celou datovou oblast, kam lze data zapisovat. Nebo naopak - tato oblast zůstane z velké části nevyužitá. V obou případech je to důvod ke změně velikosti datového objektu.

Zvětšení i zmenšení objektu jsou operace, které fungují pomocí stejného algoritmu, liší se pouze délky nových objektů. Když program zaznamená nedostatek či přebytek místa v datovém objektu, informuje uživatele a nabídne mu, zda chce provést změnu velikosti či nikoliv. Pokud uživatel zvolí, že ano, program dále pokračuje v sedmi krocích.

Prvním krokem je zkopírovaní datového objektu, který chce uživatel změnit. Tato kopie dostává vždy název *TempObj* a k jejímu vytvoření se využívá funkční blok *DatObjCopy* z knihovny *DataObj*.

Dalším krokem je vytvoření nového objektu o stejné velikosti, jakou má objekt zkopírovaný. Tento nový objekt nese vždy název *TempObj2* a k jeho vytvoření se využívá funkční blok *DatObjCreate* z knihovny *DatObj.*

Třetí krok je tím nejdůležitějším. Jeho cílem je přenést data z objektu *TempObj* do objektu *TempObj2* takovým způsobem, aby byla poté v objektu *TempObj2* seřazena za sebou. Ke kopírování jsem využil funkci *brsmemcpy* z knihovny *ArBrStr*, která umí kopírovat data o zadané délce ze zdrojové adresy na adresu cílovou. Program postupně prochází pole datových záznamů daného objektu, kde se vždy získají informace o délce a posunu v datovém objektu. Tyto informace jsou poté poskytnuty funkci *brsmemcpy* jako délka čtených dat a zdrojová adresa. Cílovou adresou je pak vždy první volné místo v objektu *TempObj2*. Tímto způsobem se překopírují všechny datové záznamy.

Po zálohování dat přichází čtvrtým krok - smazání originálního objektu. K jeho smazání je využit funkční blok *DatObjDelete* z knihovny *DataObj.*

V pátem kroku se vytvoří nový datový objekt se stejným názvem, jako měl objekt původní. Do jeho datové oblasti jsou pak zapsána data z objektu *TempObj2*. Na základě toho, zda chceme objekt zvětšit či zmenšit, bude mít nový objekt velikost dvojnásobnou či poloviční oproti objektu původnímu. K vytvoření nového datového objektu se zde opět využívá funčkní blok *DatObjCreate*.

V šestém a sedmém kroku se již pouze smažou záložní objekty *TempObj* a *TempObj2*. Ke smazání je opět využit funkční blok *DatObjDelete*.

Jestliže uživatel zvolí možnost, že nechce změnit velikost datového objektu, program tuto operaci neprovede, přejde do stavu *WAIT* a čeká na další příkazy.

5 Návod na ovládání programu

V této kapitole se vynasnažím postupně ukázat všechny možnosti programu. Případní uživatelé mohou pak tuto část mé práce považovat za návod, jakým způsobem program v prostředí Automation Studia ovládat.

5.1 Informace na úvod

Navrhl jsem speciální strukturu, která slouží jako jednoduché rozhraní pro komunikaci mezi uživatelem a logikou programu. Pomocí tohoto rozhraní uživatel program ovládá a jsou mu zpětně zasílány informace o výsledcích operací, chybových hlášeních atd. Důraz je kladen především na přehlednost a jednoduchost ovládání.

Aby bylo možné začít program v Automation Studiu používat, je nutné si nejprve zapnout *Monitor* mód a přidat si strukturu do *Watch* s názvem *userIFace*, která je umístěna v seznamu proměnných programu *user_io*.

5.2 Struktura userIFace

Celá struktura je rozdělena do pěti logických oddílů. Jsou jimy: *prgInfo*, *dataOUT*, *control*, *structures* a *objInfo.*

5.2.1 prgInfo

Tento oddíl slouží k informování uživatele o aktuálním dění programu a skládá se celkem ze čtyř proměnných: *status*, *progress*, *message*, *help*.

Proměnná *status* vyjadřuje, v jakém stavu se program zrovna nachází. Mohou nastat celkem čtyři. Prvním stavem je *WAITING*, který uživateli sděluje, že program je nečinný a čeká, až mu bude zadán nějaký příkaz. Druhým stavem je *PROCESSING*, který nám říká, že program zaznamenal příkaz od uživatele a zpracovává data. Během této operace uživatel nemůže zadávat další požadavky. Třetím stavem je S*UCCESS*. Ten informuje uživatele o úspěšném dokončení požadavku a je opět možné zadávat nové příkazy. Poslední stav - *ERROR* hlásí případný výskyt nějakého problému.

Druhá proměnná prvního oddílu se nazývá *progress* a procentuelně vyjadřuje, kolik dat již bylo zpracováno. Předposlední proměnnou je *message*. Zde se uživateli zobrazují doplňující zprávy k běhu programu či zprávy chybové. Poslední proměnná se jmenuje *help* a poskytuje uživateli nápovědu k různým situacím, které mohou v programu nastat.

5.2.2 dataOUT

Do tohoto oddílu uživatel zadává data, která program vyžaduje pro konkrétní operace. Oddíl se skládá celkem ze čtyř proměnných. První z nich je *objName*, kam uživatel zadává jméno objektu, přičemž jeho maximální délka může být deset znaků. Další proměnnou je *objLength*. Zde si uživatel volí velikost objektu v bytech. Třetí proměnná je *dataName* a sem uživatel zadává název datového záznamu, který má být zapsán do objektu. Maximální délka takového názvu může být opět deset znaků. Poslední proměnná je typu struktura a nazývá se *structType.* Uživatel si zvolí jednu z pěti možných struktur, ze které chce data zapsat nebo do které chce data načíst.

5.2.3 control

Zde jsou umístěny aktivní prvky, kterými může uživatel program ovládat. Celý oddíl je složen celkem ze dvou proměnných. První z nich se nazývá *errorHan*. Ta je typu struktura a slouží k řešení vzniklých chybových stavů. Uživatel zde má k dispozici tři možnosti, jak na různé chybové stavy reagovat. První z nich je *errorFix.* Touto volbou uživatel pouze programu sděluje, že se pokusil vyřešit daný problém, a program má jeho řešení zkontrolovat. Další dvě možnosti jsou *yes* a *no*, které logicky slouží jako odpověď typu ano a ne. Program totiž někdy zahlásí chybu a dotáže se, zda vzniklý problém chce uživatel řešit způsobem, který mu nabídne, či nikoliv. Druhou proměnnou oddílu *control* je *command*. Ta je opět typu struktura a obsahuje všechny příkazy, ze kterých si uživatel může vybírat. Příkazů je celkem pět: *create*, *read*, *write, erase a delete.*

5.2.4 structures

Tento oddíl obsahuje všechny dostupné struktury. Do struktur může uživatel zapisovat data, která chce, aby se uložila do datového objektu, anebo si do nich může nechat data z objektů načíst. Vždy podle toho, jaký typ struktury uživatel zvolí ve *structType* v oddílu *dataOUT*, bude tato konkrétní struktura aktivní. To znamená, že program při operaci čtení či při zápisu bude pracovat pouze s touto jednou strukturou.

5.2.5 objInfo

Zde jsou uživateli poskytovány informace o doposud vytvořených datových objektech a záznamech v nich uložených. Tento oddíl má celkem tři proměnné. Prvními dvěma jsou *objectCount* a *dataCount*, které informují o celkovém počtu vytvořených objektů a datových záznamů. Třetí proměnná je typu pole a jmenuje se *nameList*. Na

každém prvku tohoto pole je vždy uložen název objektu a ještě další pole. To obsahuje názvy všech v objektu uložených datových záznamů a jejich typů.

5.3 Vytvoření objektu

Aby bylo možné vytvořit nový datový objekt, je nejprve nutné vyplnit název objektu a jeho délku. Než tak ale učiníte, je dobré se podívat do seznamu existujících objektů a zkontrolovat, zda již nemáte vytvořen objekt se stejným názvem. V takovém případě by Vám program vrátil chybové hlášení. Dále v oddílu *control* zvolte příkaz *create*. Nyní si můžete všimnout, že program změnil svůj stav na *PROCESSING* a hodnota znázorňující postup programu narůstá. Jestliže jste zadali správné vstupy, program po chvíli změní stav na *SUCCESS* a zobrazí zprávu o úspěchu. Jestliže byl však některý ze vstupů zadán chybně, program změní svůj stav na *ERROR*, zobrazí chybovou zprávu a nápovědu k odstranění problému.

5.4 Smazání objektu

Pro smazání již existujícího objektu stačí pouze zadat jeho název. Opět doporučuji se nedříve podívat do seznamu objektů a zkontrolovat, jestli zadáváte správné jméno objektu, abyste se vyhnuli případným chybovým stavům. Poté již stačí zvolit příkaz *delete.* Jestliže bylo vše nastaveno v pořádku, program za chvíli změní svůj stav na *SUCCESS* a vrátí zprávu o úspěchu. V opačném případě se stav změní na *ERROR* a je zobrazena příslušná chybová zpráva.

5.5 Vytvoření datového záznamu

Pro tuto operaci je nutné vyplnit název objektu, název datového záznamu a zvolit typ struktury, která se bude do datového objektu ukládat. Nejprve se podívejte do seznamu objektů a dat, zda názvy, které zadáváte, již náhodou neexistují. Dále je potřeba vyplnit hodnoty u příslušné datové struktury, kterou jste si zvolili. Jestliže vše máte vyplněno a nechcete již hodnoty měnit, zvolte příkaz *write*. V případě, že jste zadali korektní data, program vrátí zprávu o úspěchu. V opačném případě zprávu chybovou.

5.6 Čtení datového záznamu

Abyste si mohli nějaký záznam z objektu přečíst, je potřeba vypnit název objektu, název datového záznamu, který je v něm uložený, a také typ struktury, do které chcete data načíst. Podívejte se do seznamu objektů a dat, zda zadáváte správné názvy a správný typ struktury. Poté zvolte příkaz *read*. Jestliže jste zadali všechna data dobře, program vrátí

zprávu o úspěchu a také informaci, do které struktury se Vám data načetla. Pokud jste udělali někde chybu, zobrazí program příslušnou chybovou zprávu.

5.7 Mazání datového záznamu

K této operaci je potřeba zadat název objektu a název datového záznamu. Pro ujištění, že mažete správný záznam, se opět podívejte do seznamu objektů a dat. Jestliže již nechcete zadané názvy měnit, zvolte příkaz *erase.* V případně korektních vstupních dat program vrátí zprávu o úspěchu. V opačném případě zprávu o chybě.

5.8 Chybové stavy

První případ, kdy program zahlásí chybové hlášení, nastane, jestliže jste nezadali všechna potřebá data pro provedení dané operace, nebo jestliže jste nějaká data zadali chybně. Příkladem chybných dat může být záporná velikost objektu nebo název objektu, který již existuje. V případě takové chyby musíte opravit Vámi zadaná vstupní data. Ve zprávě o chybě se můžete dozvědět, která data to přesně jsou. Pokud nevíte, jak data opravit, zkuste se podívat na nápovědu. Po opravení chybných dat musíte ještě zadat volbu *errorFix*. Poté Vám program vstupní data znovu překontroluje a vrátí buď zprávu o úspěchu, nebo další zprávu o chybě.

Dalším případem chyby může být nedostatek volného místa v objektu, nebo naopak velmi mnoho volného místa vzhledem k velikosti objektu. Ať tak či tak, program Vám zobrazí chybovou zprávu a dotáže se, zda chcete daný objekt zvětšit či zmenšit.
Zadáním volby *yes* program provede Vámi zvolenou změnu velikosti objektu a následně požadovanou operaci. Pokud zvolíte možnost *no*, program přejde do výchozího stavu a čeká na Váš další příklaz.

Závěr

Vytvořil jsem program v jazyce Strukturovaný Text pro programovatelné logické automaty, který slouží jako správce datových objektů. Program umožňuje operace jako vytváření a mazání objektů nebo čtení a zápis dat libovolných formátů v datových objektech. Podle potřeby uživatele se datové objekty mohou také zvětšovat či zmenšovat. Program tedy pokrývá veškerou funkcionalitu, která byla požadována v zadání této práce.

Program je také možné nadále vyvíjet. Obsahuje několik základních nedostatků, které by bylo dobré odstranit, nebo alespoň minimalizovat. Zásadní problém vidím v používání statických datových struktur. Z hlediska práce s pamětí je tento způsob dost neefektivní, protože po celou dobu běhu programu zbytečně zabírá paměť o celé své velikosti. Také uživatel je do jisté míry omezen, neboť po vyčerpání kapacity statických polí již není možné ukládat další objekty nebo datové záznamy.

Zde by bylo jistě vhodnější zvolit dynamický přístup, aby se počet objektů mohl zvětšovat či zmenšovat na základě potřeby uživatele. Také paměť by byla využívána mnohem efektivnějším způsobem. Zabírala by totiž vždy pouze tolik místa, kolik je vytvořených objektů a datových záznamů. Samozřejmě i tento přístup by měl svůj limit, do kterého by bylo možné alokovat další paměť. Ten by byl do jisté míry dán systémovými prostředky.

Původně jsem se snažil takový přístup implementovat, ale z důvodu vysoké časové náročnosti jsem ho musel opustit. Nejsem si také úplně jistý, zda jazyk Strukturovaný Text je tím správným jazykem pro dynamické programování. Možná by bylo vhodnější zvolit jiný programovací jazyk, například ANSI C, který má podle mého názoru v tomto směru širší možnosti.

Dalším podstatným vylepšením by bylo také dodělání grafického prostředí pro uživatele, díky kterému by se program stal daleko přehlednějším a snazším na ovládání.

Kladně bych naopak hodnotil algoritmus, který vypočítává volná místa v datové oblasti objektů. Protože funguje zcela obecně, může uživatel v rámci jednoho objektu pracovat s datovými záznamy různých velikostí, a algoritmus bude vždy správně vypočítávat, kde vznikají volná místa, a případně je slučovat.

Použitá literatura

[1] AUTOMATION, B&R. Controls - training text. Austria : [s.n.], 2008. 205 s.

[2] BERNECKER + RAINER INDUSTRIE-ELEKTRONIK GES.M. B. H. B&R Help: DataOBJ. 2013. vyd. 2013.

Přílohy

Příloha č. 1

- Bakalarska_prace_2015_Stanislav_Mares.pdf
- Bakalarska_prace_2015_Stanislav_Mares_program
    - DataObjectsManagement.rar