ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ
V PRAZE
FAKULTA ELEKTROTECHNICKÁ
KATEDRA ŘÍDICÍ TECHNIKY

SBĚR A ZPRACOVÁNÍ DAT PRŮMYSLOVÝCH APLIKACÍ

2003

Michal Houdek

Zadání

Čestné prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č.121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne ---------- ----------

podpis

Poděkování

Úvodem bych rád poděkoval vedoucímu mé diplomové práce Ing. Pavlu Burgetovi za věcné připomínky, podněty a pomoc při návrhu a realizaci systému. Rovněž děkuji Doc. Ing. Janu Bílkovi, CSc. za oponenturu k této diplomové práci.

Anotace

Tato diplomová práce se zabývá návrhem, vývojem a implementací distribuovaného systému umožňující řízení na hierarchicky vysoké úrovni, určeného k výměně, zpracování a vizualizaci dat z průmyslových aplikací řídicích systémů. Data jsou mezi jednotlivými částmi systému přenášena přes standardizované rozhraní TCP/IP (Transmission Control Protocol/Internet Protocol). Systém obsahuje část zajišťující archivaci a přístup k datům uloženým na databázovém serveru podporujícím standard SQL, jmenovitě server MySQL. Vizualizace dat je zprostředkována aplikacemi schopnými číst vizualizovaná data přes standardní rozhraní Windows DDE (Microsoft Excel, Factory Suit InTouch) či aplikacemi podporujícími rozhraní ActiveX. Přes toto rozhraní je možné nejen přistupovat k datům uvnitř systému, ale i konfigurovat systém a používat všechny možnosti nabízené systémem.

Annotation

This diploma work is engaged in the design, development and implementation of the distributed system of high hierarchic control level, dedicated to exchange, process and visualize the data obtained from the factory control system. Between parts of the system the data are transferred using standard interface TCP/IP. The system also contains a part storing the process data and accessing them from the database server MySQL. Data visualization is provided using the applications that are able to read data from standard interfaces Windows DDE (e.g. Microsoft Excel, Factory Suit InTouch) and the applications that can use interface ActiveX. Using this interface, it is possible to access the data, change system configuration and use all features offered by the system.

Obsah

strana

1. Úvod a motivace 1

1.1. Moderní řídicí systém 1

1.2. Cíl práce 1

1.3. Struktura textu 2

2. Popis struktury systému 4

2.1. Základní myšlenky 4

2.2. Popis činnosti, model komunikace 7

3. Rozhraní TCP/IP 8

3.1. Několik informací o TCP/IP obecně 8

3.2. Vývoj a historie TCP/IP 9

3.3. Vztah Internetu a TCP/IP 10

3.4. Architektura TCP/IP 10

3.4.1. Síťová vrstva 12

3.4.2. Transportní vrstva 13

3.4.3. Aplikační vrstva 14

3.5. Adresování v TCP/IP sítích 14

3.5.1. Základní pojmy 15

3.5.2. Typy IP adres 15

3.5.3. Konvence IP adres 16

3.5.4. Porovnání IP adres a fyzických adres v počítačových sítích 17

3.5.5. IPnG – výhled do budoucnosti 17

3.6. Přenos dat v sítích TCP/IP 17

3.7. Struktura dat v TCP/IP 18

3.8. Implementace TCP/IP v prostředí Borland Builder 18

3.8.1. Spojení ze strany klienta 19

3.8.2. Spojení ze strany serveru 19

4. SQL server 20

4.1. Proč MySQL, filosofie MySQL 20

4.2. Specifika a omezení MySQL 21

4.2.1. Rozdíly oproti ANSI SQL92 21

4.2.2. Transakce a MySQL 22

4.3. Základní principy v MySQL 22

4.4. Stabilita MySQL 22

4.5. ODBC 23
4.6. Databáze, ODBC a Borland C++ Builder 24
5. Implementace systému..... 26
5.1. Struktura paketů (úloh) přenášených v systému 26
5.2. Výměna proměnných v síti UniServeru (UniNetwork).....29
5.2.1. Konvence pojmenovávání proměnných 29
5.3. Server „ UniServer “.....30
5.3.1. Architektura serveru a tok dat 30
5.3.2. Nastavení serveru 31
5.3.3. Navázání komunikace, přihlášení a odhlášení 32
5.3.4. Kontrola komunikace, nastavení uživatelů 32
5.3.5. Seznam příkazů podporovaných UniServerem 33
5.4. Klient Builder komponenta, ActiveX klient 35
5.4.1. Architektura 36
5.4.2. Datové položky komponenty..... 36
5.4.3. Metody komponenty..... 37
5.4.4. Konfigurace komponenty, konfigurace v klientech 38
5.5. SQL Bridge.....39
5.5.1. Databáze, návrh a struktura..... 39
5.5.2. Architektura programu SQL Bridge 42
5.5.3. Konfigurace 42
5.6. DDE Bridge 43
5.6.1. Architektura programu DDE Bridge 44
5.6.2. Konfigurace 44
5.6.3. Připojení na data DDE Bridge 44
5.7. DDE Client.....45
5.7.1. Architektura programu DDE Klient 45
5.7.2. Konfigurace 45
5.8. OPC Bridge.....45
5.8.1. Architektura programu OPC Bridge..... 45
5.8.2. Konfigurace 46
5.9. OPC HDA Bridge 46
5.9.1. Architektura programu OPC HDA Bridge..... 47
5.9.2. Konfigurace 47
5.10. Matlab Bridge 47
5.10.1. Popis funkcí, možností a zabezpečení Matlab Bridge..... 47
5.10.2. Architektura programu Matlab Bridge 48
5.11. UniBridge.....49
5.11.1. Konfigurace 50
5.11.2. Architektura programu UniBridge 50

5.12. File Man 50
5.12.1. Architektura programu File Man 51
5.13. Test Signals 52
5.13.1. Architektura programu Test Signals 52
5.14. Data Pump 53
5.14.1. Architektura programu Data Pump 53
5.14.2. Konfigurace 53
5.14.3. Popis činnosti programu 54
5.15. Klient InOut 54
5.16. UniClient 54
5.16.1. Panel „Connection“ 55
5.16.2. Panel „Send task“ 55
5.16.3. Panel „Send task B“ 55
5.16.4. Panel „SQL Result“ 56
5.16.5. Panel „Variables“ 56
5.16.6. Panel „Variables B“ 56
5.16.7. Panel „FileMan“ 56
5.16.8. Panel „Matlab cmd“ 57
5.16.9. Panel „Matlab matrix“ 57
5.16.10. Panel „Matlab picture“ 57
5.16.11. Panel „OPC Bridge“ 57
5.16.12. Panel „OPC HDA Bridge“ 58
5.16.13. Panel „Demo“ 58
5.16.14. Panel „UniNetwork structure“ 58
5.17. UniMessenger 58
5.18. UserAccesServer 59
5.19. Klient UniChat 59
5.20. Klient Signal 59
5.21. Klient Runner 60
5.22. Klienti typu ActiveX 60
5.22.1. Objekt ActiveX poskytující připojení do sítě UniNetwork 60
5.22.2. Další ActiveX klienti 61
6. Test systému UniNetwork 62
7. Shrnutí, závěr a možnosti dalšího vývoje systému 64
7.1. Možnosti dalšího vývoje systému 64
7.2. Již realizovaná použití systému UniNetwork 66
8. Seznam literatury 67
9. Přílohy ..... A

9.1. Rozhraní TCP/IP ..... A
9.1.1. Příklady užívaných portů ..... A
9.1.2. Struktura dat v TCP/IP ..... A
9.1.3. Příklady doménových serverů ..... B
9.2. Rychlost serveru MySQL, výsledky testů ..... B
9.3. Okna a příkazy jednotlivých programů - částí systému C
9.3.1. UniServer C
9.3.2. Klient Builder komponenta, ActiveX klient ..... D
9.3.3. SQL Bridge ..... J
9.3.4. DDE Bridge ..... O
9.3.5. DDE Klient ..... P
9.3.6. OPC Bridge ..... Q
9.3.7. OPC HDA Bridge ..... S
9.3.8. Matlab Bridge V
9.3.9. UniBridge X
9.3.10. File Man ..... AA
9.3.11. Test Signals CC
9.3.12. Data Pump ..... DD
9.3.13. Klient InOut ..... EE
9.3.14. UniClient ..... FF
9.3.15. UniMessenger ..... MM
9.3.16. Klient User Access Serer ..... NN
9.3.17. Klinet UniChat ..... OO
9.3.18. Klient Signal ..... PP
9.3.19. Klient Runner ..... QQ
9.4. CD ROM ..... SS

Seznam obrázků

strana
Obrázek 2.1.1: Architektura systému, nejhrubší náhled 5
Obrázek 2.1.2: Model komunikace se dvěma servery..... 6
Obrázek 2.1.3: Model komunikace s jedním serverem..... 6
Obrázek 2.2.1: Chronologický model komunikace..... 7
Obrázek 3.6.1: Standardní model TCP aplikace 17
Obrázek 3.6.2: Standardní model UDP aplikace.....18
Obrázek 3.7.1: Struktura dat v TCP/IP rámcích 18
Obrázek 4.5.1: Funkce ODBC.....24
Obrázek 4.6.1: Architektura přístupu k databázím přes BDE.....25
Obrázek 5.1.1: Rozšíření modelu TCP/IP 28
Obrázek 5.3.1: Schéma architektury UniServeru 30
Obrázek 5.4.1: Architektura komponenty univerzálního klienta 36
Obrázek 5.5.1: Schéma architektury SQL Bridge 42
Obrázek 5.6.1: Architektura DDE Bridge 44
Obrázek 5.10.1: Architektura Malab Bridge 48
Obrázek 5.11.1: Princip UniBridge.....49
Obrázek 5.11.2: Architektura UniBridge 50
Obrázek 5.12.1: Architektura File Man 51
Obrázek 5.13.1: Architektura Test Signals.....52
Obrázek 5.14.1: Architektura Data Pump 53
Obrázek 5.22.1: Architektura klienta ActiveX 61
Obrázek 5.22.1: Schéma konfigurace systému při testu.....62
Obrázek 9.1.1: Doménový systém ..... B
Obrázek 9.3.1: Hlavní okno UniServeru C
Obrázek 9.3.2: Formulář s nastavením obsažený v komponentě ..... D
Obrázek 9.3.3: Formulář s nastavením používaný v klientech ..... D
Obrázek 9.3.4: Hlavní okno SQL Bridge .....J
Obrázek 9.3.5: Hlavní okno DDE Bridge ..... O
Obrázek 9.3.6: Hlavní okno DDE Client ..... P
Obrázek 9.3.7: Hlavní okno OPC Bridge..... Q
Obrázek 9.3.8: Hlavní okno OPC HDA Bridge ..... S
Obrázek 9.3.9: Hlavní okno Matlab Bridge..... V
Obrázek 9.3.10: Hlavní okno programu UniBridge X
Obrázek 9.3.11: Hlavní okno programu File Man ..... AA
Obrázek 9.3.12: Hlavní okno programu Test Signals.....CC
Obrázek 9.3.13: Hlavní okno programu Data Pump ..... DD
Obrázek 9.3.14: Hlavní okno programu InOut .....EE
Obrázek 9.3.15: Okna aplikace UniClient LL
Obrázek 9.3.16: Hlavní okno programu UniMessenger.....MM
Obrázek 9.3.17: Hlavní okno programu UserAccessServer .....NN
Obrázek 9.3.18: Hlavní okno programu UniChat ..... OO
Obrázek 9.3.19: Hlavní okno programu Signal.....PP

Seznam tabulek

strana
Tabulka 3.4.1: Srovnání modelů TCP/IP a RM ISO/OSI....11
Tabulka 3.4.2: Příklad modelu TCP/IP....12
Tabulka 3.5.1: Třídy IP adres....16
Tabulka 3.5.2: Desítková tečkovaná notace a IP adresy....16
Tabulka 3.5.3: Konverze síťových adres....17
Tabulka 5.1.1: Struktura paketů serveru....26
Tabulka 5.5.1: Tabulky databáze používané v SQL Bridge....40
Tabulka 5.12.1: Tabulka práv uživatelů klienta File Man....51
Tabulka 5.13.1: Seznam proměnných programu Test Signals....52
Tabulka 5.22.1: Datové toky při testu....63
Tabulka 9.1.1: Výběr standardních „wel known“ portů.... A
Tabulka 9.1.2: Struktura IP datagramu.... A
Tabulka 9.2.1: Test MySQL, čtení z databáze.... B
Tabulka 9.2.2: Test MySQL, ukládání do databáze.... C
Tabulka 9.3.1: Skript pro definici tabulek SQL Bridge.... K

Klíčová slova

TCP/IP, socket komunikace, distribuovaný řídicí systém, archivace procesních dat, komunikace, výměna dat, DDE

Key words

TCP/IP, socket connection, distributed control system, storing of process data, communication, data exchange, DDE

1. Úvod a motivace

1.1. Moderní řídicí systém

V dnešní prudce se rozvíjející době jsou na řídicí systémy kladeny čím dál větší požadavky. Zároveň se od takového moderního řídicího systému očekává značná univerzálnost použití a s tím související rozsáhlé možnosti konfigurace a nastavení systému. Nové metody řízení používají složitý matematický aparát, který je potřebný nejen během návrhu (design time), ale často i při vlastním chodu systému (run time). Obzvláště v poslední době se do řízení prosazuje používání metod počítačového rozpoznávání a virtuálního modelování řízeného výrobního procesu. Se složitostí systému úzce souvisí i nutnost kvalitní a přehledné vizualizace, která umožňuje obsluze mít dobrý přehled o stavu zařízení a jeho snadnou ovladatelnost. Tento požadavek je důležitý obzvláště při poruchách a havarijních stavech, kdy rychlé reakce operátora umožňují kvalitní identifikaci problému a jeho následné rychlé odstranění, což je velmi důležité z hlediska ekonomičnosti provozu zařízení. Dalším požadavkem na vizualizaci (a tedy i řídicí systém) je, aby umožňovala přístup k vzdáleným řídicím systémům po počítačové síti, a tak poskytovala správu systému dohledovému pracovišti a zároveň i hierarchicky vyšším úrovním řízení výroby. Jako vhodné přenosové médium k těmto účelům je Internet a to hlavně díky své rozšířenosti. Pro kvalitní a efektivní řízení je nutné mít na hierarchicky vyšších úrovních přístup k datům ze systému i z minulosti. Tento požadavek vede k tomu, že do řídicího systému je nutné zakomponování databázové podpory.

Z marketingového hlediska platí pro výrobce, že právě univerzálnost použití snižuje prodejní cenu jednotlivých kusů produktu vzhledem k výrobním nákladům. A tedy také cena vlastního řídicího systému je vázána na univerzálnost a volnost, která je poskytnuta pro designéra dané konkrétní aplikace systému. Na druhou stranu je to opět univerzálnost a tedy komplikovanost co zvyšuje finální cenu produktu, který je složitý na výrobu a jeho vývoj zabírá mnoho času a pracovních sil.

Shrnutí těchto faktů spolu se zkušenostmi získanými na studijní stáži v Belgii při spolupráci na mezinárodním projektu „Lablink – Virtual lab“, vedlo ke vzniku této práce.

1.2. Cíl práce

Cílem této diplomové páce je vytvoření systému programů s distribuovanou strukturou realizujícího výměnu dat mezi jednotlivými částmi rozsáhlého distribuovaného řídicího systému, kde jsou jednotlivé komponenty systému založeny na diametrálně rozličných podstatách. Jedná se o propojení, nadstavbu a zapouzdření dílčích řídicích systémů spolu s aplikacemi používanými v souvislosti s řídicí technikou, ale i mimo ni. Jde například o takové aplikace, které umožňují používání složitého

matematického aparátu, např. Matlab, speciálně pak využití jeho součásti „image processing toolbox“ pro zpracování obrazových informací z řídicích systémů. Nebo programy pro analýzu dat, včetně jejich historie, jako je například Microsoft Excel. Dále napojení vizualizačních nástrojů na všechna data v systému k čemuž je určen např. program InTouch z balíku programů Factory Suit 2000. Je vyžadováno, aby tyto aplikace měly přístup k procesním datům, které se nacházejí v řídicích automatech PLC a na průmyslových sběrnicích. Další neméně důležitou částí systému je vzdálená správa souborů pro tyto aplikace. Vlastní manipulace s daty technologických procesů je realizována přes servery standardu OPC, OPC HDA, standardu DDE a databázové struktury.

Mnohé z těchto specifikací a požadavků na systém vycházejí z již zmiňovaného projektu „Lablink - Virtual lab“. Částí tohoto projektu, na které jsem se účastnil, bylo zpřístupnění školního zařízení studentům, kteří se nemuseli osobně účastnit svých cvičení, ale mohli všechna potřebná zařízení a k nim příslušející programové vybavení ovládat na dálku z domova po Internetu. Tento systém vyžaduje, aby všechny části zpracovávané úlohy již byly mezi sebou propojeny a nakonfigurovány a student pouze vytvářel řídicí programy realizující požadovanou činnost. Tyto své programy poté nahrál do cílových zařízení a aplikací a zavedl je do procesu řízení. Podrobnější informace o celém projektu lze nalézt například na Internetové adrese [http://www.iwt-kdg.be/Lablink/virtuallabs.htm, nebo na školním webu http://www.cvut.cz/fee/k317/projekty/lablink/usefullsites.html]. Některé z požadavků z projektu „Lablink - Virtual lab“ byly rozšířeny a doplněny a staly se částí zadání této diplomové práce.

Dalšími požadavky, které byly kladeny na tento řídicí systém jsou obsáhnutí široké oblasti použití a možnost nakonfigurovat jej přesně dle potřeb daného problému. Také byl kladen důraz na efektivní využití paměti počítače při zpracování dat a to zejména při jejich skladování v databázích.

Jak již bylo uvedeno výše, jako rozhraní pro přenos dat mezi částmi systému je použito rozhraní TCP/IP. To je ze své filozofie nezávislé na přenosovém mediu a je určeno jak pro WAN tak i pro LAN, jak pro sériové linky, koaxiály, tak i pro vysokorychlostní optické sítě. Použití sítě s těmito vlastnostmi umožňuje velmi široké možnosti nasazení systému.

Systém celkově uspořádaný s takovouto architekturou a filosofií umožňuje realizovat složité a efektivní řízení výrobních procesů na velmi vysoké úrovni hierarchie řízení.

Další motivací pro návrh a realizaci popisovaného projektu je skutečnost, že systém propojující dílčí části, založené na takovýchto rozličných bázích, splňující výše uvedené požadavky, v běžně používaných programových balících víceméně neexistuje.

1.3. Struktura textu

V kapitole 2 bude práce pojednávat o struktuře celého navrhovaného systému, o jeho rozdělení a základních principech systému. Budou zde

uvedeny základní myšlenky a údaje o komunikaci v Internetu po protokolu TCP/IP v kapitole 3. Další kapitola, kapitola 4, pojednává o databázovém serveru MySQL, o jeho výhodách a nevýhodách, o možnostech, které nabízí a o připojení přes ODBC. Dále bude popsána implementace systému s popisem jednotlivých rozhraní částí systému v kapitole 5. Postup ověření funkčnosti a jeho výsledek je uveden v kapitole 6. Návrhy a zamyšlení nad možným budoucím vývojem systému budou rozebírány v kapitole 7 spolu se shrnutím výsledků a závěrem celé práce.

Aby bylo učiněno za dost korektnímu uvedení čtenáře do tohoto textu, vysvětluji na tomto místě také četné používání anglických či počeštěných anglických výrazů. Podle mého názoru je doslovný překlad některých odborných slov používaných v technické praxi v celkovém kontextu spíše matoucí a odvádí pozornost od důležitějších myšlenek, které jsou v textu a vyžadují koncentrovanost. Dalším důvodem je i to, že vlastní téma této práce předpokládá čtenáře působícího v technických sférách, kde angličtina je běžně používaný jazyk a proto vyjadřování se tímto způsobem by nemělo působit problémy.

Protože se v systému objevuje více typů serverů, klientů a přenosových sítí, mohlo by snadno dojít ke zmatení čtenáře. Budu se proto snažit v dalším textu pokaždé přesně specifikovat o jaký konkrétní význam slova jde. Toto časté používání přívlastků u těchto jmen v textu není nadbytečné. Snažím se o popis, který by byl zřejmý a srozumitelný bez nutnosti mnohanásobného se vracení a pátrání po přesném kontextu, které unavuje a znesnadňuje čtení a orientaci v textu.

2. Popis struktury systému

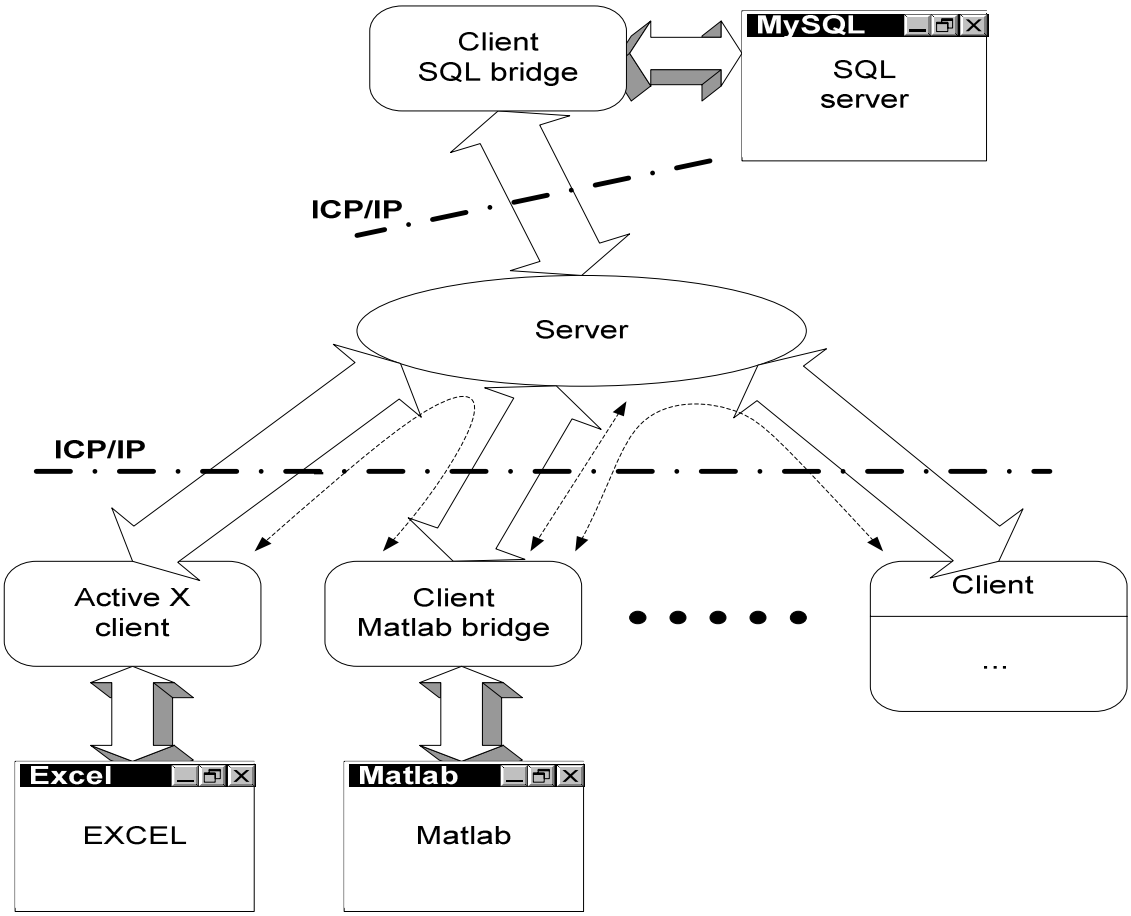
V této kapitole se budu zabývat základními myšlenkami podle kterých byla budována architektura systému (kapitola 2.1). Uvedu zde model komunikace v systému a popíši činnosti systému v kapitole 2.2. Také zde bude vysvětlena struktura paketů dat přenášených přes TCP/IP rozhraní systému (kapitola 5.1).

2.1. Základní myšlenky

Pří návrhu struktury systému jsem vycházel ze základních požadavků, kterými jsou univerzálnost použití, možnosti konfigurace systému tak, aby odpovídal přesně řešenému problému. A požadavku možnosti distribuovat přístup k datům a konfiguraci systému po Internetu. Z těchto hledisek je pro systém nejvýhodnější použití struktury typu server – client.

Zvolil jsem řešení, kdy v celém systému je jeden hlavní server, který obstarává všechnu komunikaci. Části systému typu klient jsou pak jednotlivá zařízení, programy, periferie či jiné systémy, které chceme do výměny dat zapojit. V tomto uspořádání je na server kladeno několik hlavních úkolů. Zaprvé server musí zabezpečit co možná nejrychlejší výměnu dat s připojenými klienty. Dále musí obstarávat připojování klientů. Pro zachování bezpečnosti musí kontrolovat uživatele, identifikované pomocí jména a hesla, kteří se přihlašují do systému a jejich uživatelská práva. A v neposlední řade musí být stabilní, přesněji řečeno musí vykazovat očekávané a standardizované chování v neočekávaných situacích. Souhrnem lze říci, že server, který se ideálně hodí pro tyto účely, by měl být co nejjednodušší a co nejrychlejší.

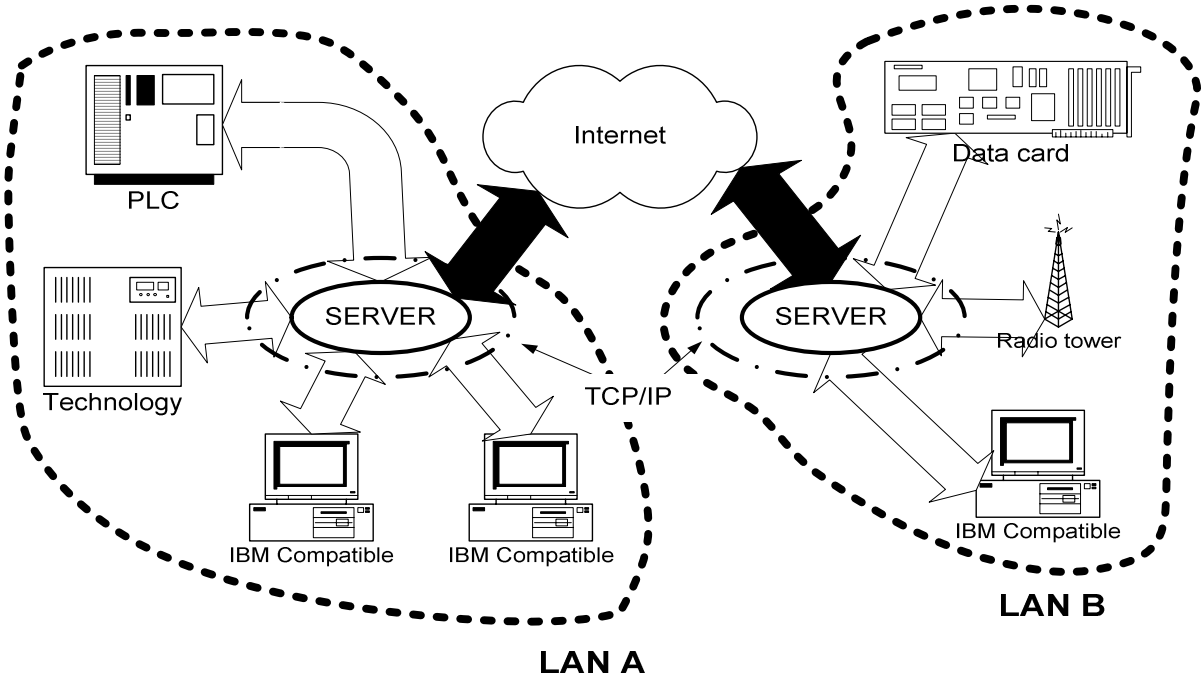
Na obrázku 2.1.1 je čárkovanými šipkami naznačena komunikace, která v systému může probíhat. Je to komunikace jednak mezi klientem a serverem a hlavní komunikace, tedy komunikace mezi dvěma klienty. Ta se ve skutečnosti sestává ze dvou komunikací mezi serverem a klientem, kdy předávaná data jsou až na některé položky v hlavičce paketu totožná. Komunikace, kdy posílaná data jsou určena pouze pro server, se vždy týká správy připojení daného klienta (který posílá data) do systému. Muže jít o požadavek navázání či zrušení komunikace, nebo o dotazy na stav systému a seznam připojených klientů.



Obrázek 2.1.1: Architektura systému, nejhrubší náhled

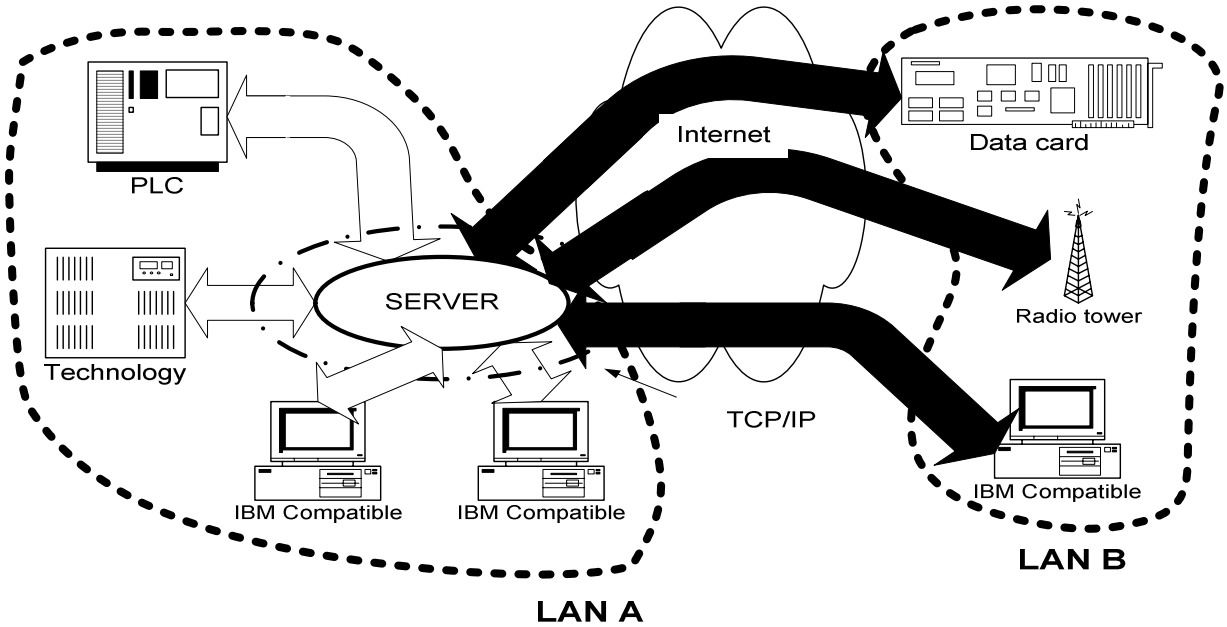
Takový server by měl poskytovat jednoduchý protokol TCP/IP rozhraní, aby bylo možné jednoduché napojení z různých klientů, kteří mohou být založeny na různých platformách. Což nemusí být jen operační systémy stolních počítačů třídy PC, ale jakékoliv zařízení, které má přístup do sítě Internet a je schopné provozování socketového spojení. Jako příklad může být řídicí automat PLC, nebo jednočip mající síťové připojení.

Systém by měl být univerzální do té míry, aby bylo možné do systému zapojit více serverů a ty vzájemně propojovat. To je důležité zejména pro urychlení komunikace mezi jednotlivými klienty a zajištění daného časového limitu na doručení zprávy k adresátovi. Jako příklad uveďme situaci, kdy máme spojit šest klientů, kdy tři se nalézají v jedné lokální síti LAN a další tři v jiné vzdálené síti LAN. O propojení těchto dvou sítí předem víme, že je pomalé a nemuselo by stačit na požadovanou rychlost výměny dat. Řekněme, že jde o dva samostatné řídicí systémy nějakých technologických procesů, které se mají navzájem synchronizovat v počtu vyrobených kusů, ale co nejméně omezovat. Pak vyžadujeme, aby řídicí systém jednotlivých technologii měl možnost uvnitř komunikovat s vyšší rychlostí než jaká je vyžadována pro komunikaci mezi těmito dvěma systémy. Uspořádání systému, které řeší tento problém je na dalším obrázku.



Obrázek 2.1.2: Model komunikace se dvěma servery

Na obrázku vidíme, že do systému jsou napojeny dva servery, které jsou vzájemně propojeny přes síť Internet. Takováto konfigurace umožní výměnu dat mezi jednotlivými klienty obou serverů, pokud je tak adresováno, jen v rámci lokální sítě LAN. A tudíž o mnoho rychleji, než pokud by tomu bylo v případě použití jen jednoho serveru, jak ukazuje následující obrázek.



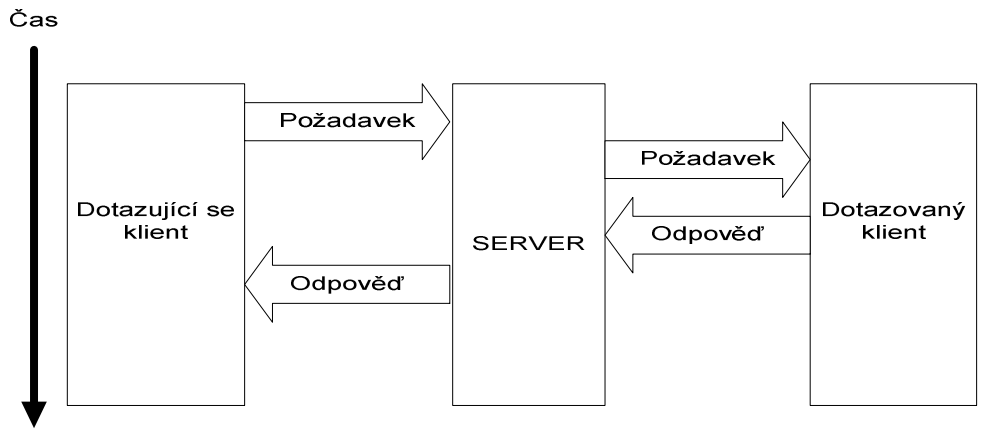
Obrázek 2.1.3: Model komunikace s jedním serverem

V uspořádání s jedním serverem se všechna data mezi klienty fyzicky napojenými uvnitř jedné sítě LAN B musejí přenášet přes Internet, dokonce dvakrát, což značně (řádově) zpomaluje výměnu dat. Takovéto uspořádání je pro časově náročnější aplikace nevhodné.

2.2. Popis činnosti, model komunikace

Při konstrukci modelu komunikace a pohybu dat jsem kladl důraz na univerzálnost dat přenášených v systému a jednoduchost struktury těchto dat. Jako nejlepší řešení se ukázalo být použití paketové komunikace. Všechny události, požadavky, odpovědí a vůbec veškerá komunikace v systému probíhá formou paketů o pevně definované struktuře (viz níže v kapitole 5.1). Obsahem těchto paketů mohou být naprosto libovolná data. Jediné co musí být splněno je aby cílový klient paketu dokázal těmto datům porozumět a správně je interpretovat. Jednotlivé pakety pak mají význam jakýchsi příkazů nebo úkolů či jen informační charakter. Dále o těchto paketech budu mluvit jako o úlohách (dále v textu je v názvech procedur a vlastností často použit anglický překlad – task).

Myšlenkou komunikace uvnitř celého systému je, že data - úkoly se v systému přenášejí pouze, když dojde k nějaké významné změně v systému, když si nějaký klient data vyžádá od jiného, nebo chce předat svá data někomu jinému. Pokud žádná takováto situace nenastala, komunikace v systému je nečinná. V takovémto případě se jen jednou za čas provede náhodný „PING“ na klienta, který se již po delší dobu neúčastnil komunikace. To se provádí ze dvou důvodů. Jedním z nich je přezkoušení funkčnosti komunikace. Pokud by kdykoli při přenosu došlo k výpadku dat, potažmo spojení, transportní vrstva TCP se pokusí o nápravu. Pokud bude pokus neúspěšný, vyvolá se chyba (více kapitola 3.4.2). Druhý důvod je otestování, zda se daný klient nedostal do nějakého nedefinovaného stavu a je schopen odpovídat správně na požadavky od ostatních klientů. Průběh komunikace, znázorněný v čase je zobrazen na dalším obrázku.



Obrázek 2.2.1: Chronologický model komunikace

Hlavní princip komunikace v celém systému tedy je, že každý klient muže jakémukoliv klientu zaslat příkaz (požadavek) a očekává od něj odpověď s výsledkem.

Ze samotné podstaty komunikace v Internetu, tedy paketových sítí, plyne, že data není možné předávat v reálném čase. Při vyslání jakékoliv informace do sítě Internet není zaručen čas v jakém adresát data obdrží. Blíže se o této tématice zmiňuje [4] a [9], také tato práce dále v kapitole 3. Ke všem datům, u kterých je to vyžadováno, je nutné doplnit i záznam o čase ve kterém byly získány. Toto opatření však neřeší problém, když požadovaná data v určitém čase nejsou zrovna dostupná, zpozdila se na cestě. Je tedy nutné v dalších úvahách a konstrukcích tento fakt uvažovat a používané algoritmy tomu přizpůsobit. Jediné co je možné stanovit, je jakýsi průměrný čas, ve kterém se daří informacím z protějšího konce spojení dorazit. Ale tato informace je určena pouze pro dané zatížení sítě a muže se v čase velmi významně měnit. Z dlouhodobějšího pohledu nelze s výsledky takovéhoto měření počítat.

3. Rozhraní TCP/IP

Nyní zde uvedu základní informace o síťovém rozhraní TCP/IP. Nejprve to bude několik poznámek o vzniku a terminologii. Dále uvedu něco o historickém a budoucím vývoji, o vztahu TCP/IP a Internet. Popíši architekturu a filosofii této technologie a popíši blíže vlastní pohyb dat v tomto protokolu. Další podrobnosti o TCP/IP naleznete v příloze 9.1

3.1. Několik informací o TCP/IP obecně

Technologie TCP/IP je jednou z nejúspěšnějších technologii ve světě vůbec. Je to možná důsledkem toho, že již od samého počátku byla tato technologie projektována komerčními laboratořemi, které byly zpočátku úplně a později z větší části, financovány z vojenských zdrojů.

Často se setkáváme se slovním spojením „protokol TCP/IP“, ale ve skutečnosti TCP/IP je souhrnné označení mnoha jednotlivých protokolů, které spolu úzce souvisí. V současné době je jich již více než stopadesát. Nejznámější z nich jsou právě TCP (Transmission Control Protocol) a IP (Internet Protocol). Je tedy správné mluvit o celé rodině či soustavě protokolů. V angličtině se používá označení „TCP/IP protokol suite“. Z celkového hlediska je TCP/IP mnohem víc než soustavou vzájemně propojených protokolů, jde o ucelený názor, o soustavu myšlenek a vizí, jak by počítačové sítě měly být budovány a jak by měly fungovat.

3.2. Vývoj a historie TCP/IP

Vývoj TCP/IP dosti úzce souvisí s historií Internetu. Internet sám o sobě vznikal jako experiment amerického ministerstva obrany, který měl ověřit možnost vybudování počítačové sítě propojující místa velení a schopnou přežít i jaderný úder nepřítele. Ministerstvo obrany USA si za tímto účelem nechalo vypracovat studii (od firmy Rand Corporation), aby vědělo kterým směrem se vůbec vydat. Zmíněná studie dospěla k závěru, že jedinou možností je nevytvářet v síti žádný centrální prvek - který by jistě byl prvním cílem nepřítele - a pak také nutnost předem počítat s nespolehlivostí přenosových cest, které může nepřítel kdykoli „odstřelit", ale síť jako celek by to měla přežít. Důsledkem pak bylo i zjištění, že „klasické" přenosové technologie na bázi přepojování okruhů, v tehdejší době zcela dominující ve světě spojů, nejsou pro daný účel použitelné (když nepřítel „odstřelí" část sítě, přeruší tím již zřízený okruh).

Přednost proto dostala myšlenka přepojování paketů, pocházející právě od zmíněné Rand Corporation - ale její životaschopnost bylo třeba nejprve ověřit v praxi, protože šlo o věc velmi novou a tudíž dosud příliš neimplementovanou. Americké ministerstvo obrany spojilo příjemné s užitečným, a začalo budovat rozlehlou počítačovou síť, schopnou ověřit životaschopnost přepojování paketů - a současně s tím schopnou propojit výzkumná střediska, které totéž ministerstvo financovalo ze svých prostředků na rozvoj vědy. Světlo světa tak spatřila síť ARPANET, pojmenovaná po grantové agentuře ARPA (Advanced Research Projects Agency) - což byla účelová organizace ministerstva obrany USA (DoD, Department of Defense), skrz kterou „tekly peníze" od vojáků do akademické sféry, která výzkum skutečně zajišťovala.

Původní ARPANET byl tedy vybudován na technologii přepojování paketů - ovšem takové, která byla jen narychlo „spíchnuta" pro potřeby ověření životaschopnosti celé koncepce. Toto ověření se podařilo, ale původní implementace se ukázala jako nepříliš vhodná pro praktické použití. No a tak znovu nastoupil měšec amerického ministerstva obrany, které skrze svou grantovou agenturu ARPA financovalo vývoj „řádné" a neuspěchané verze síťových protokolů na principu přepojování paketů. Tak se v průběhu 70. let zrodily protokoly TCP/IP. Časem (postupně cca od roku 1980, definitivně k 1.1.1983) na ně přešla i síť ARPANET, která postupně přerostla až v dnešní Internet.

K dalšímu prosazení protokolů TCP/IP, i mimo samotnou síť ARPANET, byl velmi důležitý krok amerického resortu obrany, za jehož peníze byly protokoly TCP/IP vyvinuty. Byly investovány další peníze do toho, aby protokoly TCP/IP byly implementovány do prostředí vhodného operačního systému. Grantová agentura ARPA zaplatila komerční firmě BBN (Bolt. Beranek and Newman, která také „stavěla" začínající ARPANET), aby tato implementovala protokoly TCP/IP v BSD Unixu.

Agentura ARPA (posléze přejmenovaná na DARPA) pak financovala i další vývoj protokolů TCP/IP. Po věcné stránce jej měla na starosti řídicí skupina s názvem ICCB (Internet Configuration and Control Board), které zpočátku předsedal duchovní otec celé koncepce TCP/IP, pan Vinton G. Cerf. Později (s odchodem pana Cerfa do komerční sféry) byla tato skupina přejmenována na IAB (Internet Activities Board), ale svou roli zastřešujícího orgánu pro další vývoj TCP/IP si podržela až do dnešních dnů, kdy sice má stále stejnou zkratku IAB, ale poněkud pozměněný název (Internet Architecture Board). Není již také podřízena agentuře ARPA a financována z jejích prostředků, ale je součástí prestižní Internet Society („společnosti pro Internet", založené v roce 1992).

Dnes již soustava protokolů TCP/IP je nezávislá na americkém ministerstvu obrany a vyvíjí se zcela odděleně, převážně v civilním sektoru. V současné době se na vývoji protokolů podílí jednotlivci sdružovaní spolkem IETF (Internet Engineering Task Force), akademické skupiny a spousta komerčních firem. V standardní soustavě TCP/IP se nachází mnoho protokolů, které původem pocházejí od komerčních firem, které realizovaly své vlastní projekty jejichž řešení se v praxi osvědčilo a ukázalo se přínosem. To jsou např. protokol NFS pro sdílení souborů (firma Sun) a zvláště pak celá koncepce WWW s příslušejícími protokoly (vyvinuté střediskem CERN).

3.3. Vztah Internetu a TCP/IP

Soustava protokolů TCP/IP byla v podstatě od začátku budována tak, aby uspokojovala potřeby prudce se rozvíjejícího Internetu. Z tohoto důvodu se často mluví o protokolech TCP/IP jako o protokolech Internetu. Avšak není zcela korektní používat Internet a TCP/IP jako ekvivalentní výrazy. TCP/IP je soustava doporučení a standardů a Internet je již zcela konkrétní realizovaná rozlehlá síť skládající se z mnoha menších nezávislých částí. Je ale třeba si uvědomit, že technologie TCP/IP může být, a mnohde je, implementována ve zcela odlišných sítích od Internetu.

3.4. Architektura TCP/IP

Síťový model TCP/IP je vystavěn na jiných předpokladech, než z jakých vychází referenční model ISO/OSI. Především z tohoto faktu pak vyplývají základní vlastnosti modelu TCP/IP a jeho odlišnosti od referenčního modelu.

Podobně jako referenční model, vychází i síťový model TCP/IP ze stejné výchozí představy o tom, že by síťové funkce měly být rozděleny do hierarchicky uspořádaných vrstev. I síťový model TCP/IP je tedy modelem vrstvovým, stejně jako RM ISO/OSI. Autoři TCP/IP však při svém počátečním rozhodování o počtu vrstev a jejich úloze vyšli z jiných předpokladů než autoři RM ISO/OSI, a proto také dospěli k odlišným závěrům, a mj. i k odlišnému počtu vrstev - ke čtyřem, zatímco referenční model ISO/OSI jich předpokládá sedm.

Jedním ze základních požadavků, které byly nastoleny na samém začátku prací na celkové koncepci TCP/IP, byl požadavek na možnost vzájemného propojení různých sítí. Tedy požadavek na to, aby prostřednictvím protokolů TCP/IP bylo možné vzájemně propojit i takové sítě, které mohly být vybudovány i na dosti odlišných principech a základních přenosových technologiích (tedy například sítě Ethernet, Token Ring, dnes FDDI, ATM, sítě s dvoubodovými spoji atd.).

Po protokolech TCP/IP se tedy od začátku požadovalo, aby „běhaly" co možná nejlépe nad nejrůznějšími protokoly nejnižších vrstev (které v rámci referenčního modelu ISO/OSI spadají do fyzické a linkové vrstvy), a také aby se apriorně neuzavíraly možnosti využít i takové přenosové protokoly, které budou teprve v budoucnu vyvinuty. Takovýto požadavek přitom v sobě nutně nesl i akceptování faktu, že životaschopné (a uživateli skutečně používané) přenosové technologie mohou vznikat i mimo rámec světa TCP/IP. A také, že není nutné znovu vynalézat již jednou vynalezené, ale že stačí již existující přenosové technologie pouze využít. Autoři RM ISO/OSI ve stejné situaci použili opačný přístup - považovali za nezbytné sami vyvinout i veškeré přenosové technologie, nebo alespoň formálně převzít a vydat jako svůj standard takové technologie, které přece jen vznikly někde jinde (to je třeba případ Ethernetu a dalších technologií, pocházejících od společnosti IEEE - standardy IEEE řady 802 byly vydány současně i jako standardy ISO 8802). K tomu ale samozřejmě referenční model ISO/OSI potřeboval fyzickou a linkovou vrstvu, zatímco síťový model TCP/IP tyto vrstvy nepotřebuje a nemá.

<table>
<tr><td rowspan="3">aplikační vrstva</td><td rowspan="3"></td><td>aplikační vrstva</td></tr>
<tr><td>presentační vrstva</td></tr>
<tr><td>relační vrstva</td></tr>
<tr><td>transportní vrstva</td><td></td><td>transportní vrstva</td></tr>
<tr><td>síťová vrstva</td><td></td><td>síťová vrstva</td></tr>
<tr><td rowspan="2">vrstva síťového rozhraní</td><td rowspan="2"></td><td>linková vrstva</td></tr>
<tr><td>fyzická vrstva</td></tr>
<tr><td>TCP/IP</td><td></td><td>RM ISO/OSI</td></tr>
</table>

Tabulka 3.4.1: Srovnání modelů TCP/IP a RM ISO/OSI

Pro přesnost a formální správnost však dodejme, že síťový model TCP/IP přece jen má nejnižší vrstvu, která svým „dosahem" pokrývá současně fyzickou i linkovou vrstvu referenčního modelu ISO/OSI. Je to vrstva označovaná jako vrstva síťového rozhraní (network interface layer). Důležité je ale to, že TCP/IP tuto vrstvu ponechává prázdnou, a sám ji nijak „nezabydluje“ konkrétními protokoly, nýbrž předpokládá, že na úrovni této vrstvy budou použita řešení vyvinutá mimo rámec TCP/IP.

<table>
<tr><th>OSI</th><th>Vrstva</th><th colspan="3">Protokoly v TCP/IP (příklady)</th></tr>
<tr><td>7</td><td rowspan="3">aplikační</td><td rowspan="3">FTP</td><td>NFS</td><td rowspan="4"></td></tr>
<tr><td>6</td><td>XDR</td></tr>
<tr><td>5</td><td>RPC</td></tr>
<tr><td>4</td><td>transportní</td><td>TCP</td><td>UDP</td><td rowspan="2">ICMP</td></tr>
<tr><td>3</td><td>internet</td><td colspan="2">IP</td><td>ARP</td></tr>
<tr><td>2</td><td>network interface</td><td colspan="3">Ethernet, FDDI, ATM, SLIP, X.25</td></tr>
<tr><td>1</td><td>hardware</td><td colspan="3">-</td></tr>
</table>

Tabulka 3.4.2: Příklad modelu TCP/IP

Síťový model TCP/IP tedy nijak nevymezuje, jaká konkrétní přenosová technologie bude použita na úrovni vrstvy síťového rozhraní a v důsledku toho pak ale nemůže sám nic předpokládat o vlastnostech této technologie. Nemůže tedy dopředu počítat s tím, že půjde o takovou technologii, která je relativně spolehlivá a nedochází u ní ke ztrátám či poškození dat příliš často, nebo že naopak jde o přenosovou technologii, která ztrácí data častěji. Podobně nemůže předpokládat nic ani o dalších vlastnostech, jako třeba o rychlosti přenosu, přenosovém zpoždění, maximální velikosti přenášených bloků apod.

Na druhé straně však má TCP/IP za úkol nabídnout (na úrovni vyšších vrstev než je vrstva síťového rozhraní) stejné možnosti, podmínky i stejný způsob práce - tedy vlastně zakrýt případná specifika konkrétních síťových technologií a vytvořit nad nimi jednotné prostředí, nabízející jednotné služby, jednotný způsob adresování apod.

3.4.1. Síťová vrstva

Síťová vrstva by se měla soustředit především na co možná nejrychlejší přenos dat, a nikoli na zajištění spolehlivosti a případných dalších „bohatých" funkcí. Filosofie, ze které tato vrstva vychází je, že spolehlivost si lépe a efektivněji zajistí ti, kdo ji budou skutečně potřebovat. Tedy až vyšší vrstvy, případně až jednotlivé aplikace. Na úrovni síťové vrstvy je tedy implementovaný protokol, který funguje jako nespolehlivý. Sice se snaží o bezchybný přenos, ale když se mu to nepodaří a někde se něco ztratí či poškodí, nepovažuje za svou povinnost postarat se o nápravu a místo toho očekává, že o případnou nápravu se postarají vyšší vrstvy.

Zmíněný protokol dostal jméno Internet Protocol (zkratkou IP), což nesouvisí ani tak s Internetem jako takovým (který v době vzniku TCP/IP byl ještě zárodečným Arpanetem), jako spíše se skutečností, že má na starosti vzájemné propojení jednotlivých dílcích sítí (obecný „internet", s malým „i").

Dalším charakteristickým rysem protokolu IP je jeho nespojovaný charakter, tedy při přenosu se dat nepočítá s apriorním navázáním spojení mezi odesilatelem a příjemcem, a místo toho posílá všechna data „naslepo", obdobně jako jsou odesílány jednotlivé dopisy běžnou listovní poštou. A pak

mluvíme o přenášených blocích dat jako o datagramech (IP datagramech) a o nespojovaném způsobu přenosu jako o datagramové službě.

Motivaci pro nespojovaný charakter protokolu IP je nutné hledat v celkovém požadavku na robustnost protokolů TCP/IP a jejich schopnost přežít nenadálé výpadky sítě. Dojde-li totiž někde k náhlému přerušení nějaké přenosové cesty, přenos spojovaného charakteru to ovlivní zásadním způsobem (a je nutné provést složitější nápravné akce, vrcholící navázáním nového spojení). Naproti tomu přenos nespojovaného charakteru to ovlivní mnohem méně, zde je každý jednotlivý blok dat (datagram) přenášen samostatně, svou vlastní cestou, a tak jsou v případě výpadku jedné možné cesty další datagramy snadno směrovány jinudy.

3.4.2. Transportní vrstva

Zajištění spolehlivosti by mělo být spíše záležitostí koncových uzlů, než přenosové části sítě, a mělo by se tedy odehrávat na úrovni vyšších vrstev než je vrstva síťová, která musí být implementována ve všech částech přenosové sítě, zatímco vyšší vrstvy se nacházejí již jen v koncových uzlech. Ale ani do této vrstvy by nebylo zcela optimální implementovat zajišťování spolehlivosti. Aplikace mohou mít různorodé požadavky na přenosové služby. Některé budou spolehlivost skutečně požadovat, zatímco jiné aplikace dají přednost rychlejšímu a pravidelnějšímu přenosu dat před spolehlivostí přenosu (protože fungování mechanismů pro zajištění spolehlivosti způsobuje celkové zpomalení a zavádí nerovnoměrnosti v doručování dat).

Dále je nutné si uvědomit, že mechanismy zajišťující spolehlivost nemohou mít nikdy absolutní účinnost a efektivnost, protože absolutně účinné a efektivní nejsou ani mechanismy detekce chyb (umožňující vůbec rozpoznat, že k nějaké chybě došlo). Proto každá spolehlivost je vždy relativní. Může se sice dosti těsně blížit absolutní hranici 100%, ale nikdy se k ní fakticky nedostane. V praxi by se pak mohlo stát, že transportní vrstva by sice zajišťovala spolehlivost přenosu, ale pro určitou aplikaci ne dostatečnou, a tato aplikace by si proto musela zajišťovat potřebnou míru spolehlivosti sama a znovu. Pak by ale bylo zbytečné a neefektivní, aby spolehlivost zajišťovala současně i vrstva transportní.

Z těchto důvodů poskytuje transportní vrstva volbu uživateli a implementuje dva hlavní transportní protokoly. Jeden je protokol TCP (Transport Control Protocol), který zajišťuje určitou míru spolehlivosti a druhý UDP (User Datagram Protocol), který nikoli. Vyšší vrstvy si pak mohou samy vybrat, který z těchto transportních protokolů budou využívat.

3.4.3. Aplikační vrstva

Aplikační vrstva je nejvyšší vrstva architektury TCP/IP. V této vrstvě jsou již skutečně provozovány jednotlivé aplikační programy. Ty, narozdíl od RM OSI, komunikují přímo s transportní vrstvou. Případné prezentační a relační služby si aplikační programy zajišťují samy. Aplikační vrstva zajišťuje přenos a srozumitelnost zpráv.

Základní služby této vrstvy jsou:

- TELNET , což je vlastně emulátor terminálu. Umožňuje pracovat na vzdáleném počítači tak, jako by to byl terminál nebo přímo váš počítač. Využívá služby TCP.
- FTP (File Transport Protocol) - umožňuje přenášet soubory ze vzdálených disků (virtuálně je přiřadí k počítači). Využívá TCP. Jeho klonem je TFTP - trivial FTP - u UDP protokolu.
- SMTP (Simple Mail Transfer Protocol) - využívá se pro E-mail se 7mi bitovým ASCII
- ARCHIE
- GOPHER
- WWW
- RIP (Routing Information Protocol) směrovač, který předává informace o směrování. Podrobnosti viz IPX. (od r. 1998 RIP2, která přidává ke směrování ve vrstvě 2 bezpečnost).
- DNS (Domain Name Server) – konvertuje symbolické zápisy adres na skutečnou číselnou IP adresu.

3.5. Adresování v TCP/IP sítích

Vznikla potřeba překrýt všechny rozličné způsoby adresování jednotlivých uzlů i celých sítí, vyplývající z použití příslušných přenosových technologií, a snaha používat na jednotné úrovni (tj. protokolu IP) jednotný způsob adresování, nezávislý na konkrétním druhu sítě, tzv. IP adresy. Dále je potřeba na určité úrovni abstrakce tyto dílčí sítě uvažovat jako dále nedělitelné celky (například pro potřeby tzv. směrování), zatímco v jiných situacích rozlišovat i jednotlivé uzly v rámci těchto sítí. To vedlo k nutnosti zvolit takové IP adresy, které by měly dvě logické složky - číslo, resp. *adresu sítě* jako celku, a dále *adresu uzlu* v rámci dané sítě. Pro praktickou manipulaci s IP adresami bylo samozřejmě velmi žádoucí, aby "fyzicky" byly jednosložkové (tj. byly jedinou lineární posloupností bitů), a jejich rozdělení na dvě logické složky bylo až záležitostí interpretace jejich významu.

3.5.1. Základní pojmy

IP adresa: 32 bitové číslo
pro člověka nevhodné, málo mnemonické
používá se konvence pro symbolické zapisování těchto adres
Port: 16ti bitové číslo identifikující jeden konec navázaného spojení tj. aplikaci, proces, který má zpracovávat příchozí pakety
source port – identifikace navazovatele (>1024)
– port spojení přiděluje lokální systém
destination port – musí být znám
Socket: dvojice [IP adresa, port]

3.5.2. Typy IP adres

Snaha zbytečně neplýtvat 32 bity na jednu IP adresu vedla k myšlence přizpůsobit mechanismus adresování předpokladu, že budou existovat jak hodně velké sítě (kterých ale zřejmě nebude mnoho), tak i sítě středně velké, a stejně tak sítě velmi malé (kterých zase může existovat více). Proto bylo rozhodnuto použít pružnou "hranici" mezi oběma logickými složkami adres. Zvolilo se takové řešení, které připouští tři různé formáty adres - lišící se v tom, kolik ze zmíněných 32 bitů vyhrazují pro adresu sítě jako takové, a kolik pro adresu uzlu v rámci dané sítě:

- **adresy třídy A** (Class A) jsou určeny pro velmi velké sítě: předpokládají, že pro adresu uzlu v rámci dané sítě je využito 24 nejnižších bitů celé 32-bitové adresy, což znamená, že příslušné sítě mohou mít až 2^{48} uzlů. Pro adresu sítě jako takové je pak vyhrazeno zbývajících 8 bitů, z nichž ale jeden bit je využit pro indikaci skutečnosti, že se jedná právě o tuto adresu třídy A. Takže pro adresu sítě jako celku zbývá pouze 7 bitů, neboli sítí s adresou třídy A může být jen 2^7, tj. 128.
- **adresy třídy B** (Class B) jsou určeny pro středně velké sítě. 32 bitů, které mají k dispozici, rozdělují na dvě stejně velké části: pro adresu uzlu v rámci sítě je vyhrazeno 16 bitů, což připouští až 2^{16} uzlů (65536, ve skutečnosti ale jen 65534 uzlů). Pro adresu sítě je pak vyhrazeno 14 bitů (neboť dva jsou využity pro identifikaci skutečnosti, že jde o adresu třídy B), tedy sítí s adresou třídy B může být až 2^{14}, neboli 16 384.
- **adresy třídy C** (Class C) jsou určeny pro malé sítě, které nemají více než 254 (256 - 2) uzlů - pro adresu uzlu totiž používají jen 8 z celkových 32 bitů, zatímco pro adresu sítě používají 21 bitů. Takovýchto sítí tedy může být velké množství – 2^{21}.

Kromě těchto tří základních tříd IP adres existují ještě dvě další třídy - D a E. Třída D (začínající čtveřicí bitů 1110) je určena pro tzv. skupinové

vysílání (multicasting), a třída E (začínající pěticí bitů 11110) je vyhrazena pro budoucí využití.

Třída	Tvar adresy (1.byte, 2.byte, 3.byte, 4.byte)	Začátek adresy	Počet sítí	Počet strojů
A	0 net host	1-126	126	~16 M
B	10 net host	128-191	~16 k	~64 k
C	110 net host	192-223	~2 M	254
D	1110 net	224-239	multicast	
E	1111	240-255	IP next generation	

Tabulka 3.5.1: Třídy IP adres

3.5.3. Konvence IP adres

32-bitové IP adresy je samozřejmě možné vyjadřovat jako celá dvojková čísla. Pro člověka to však není příliš srozumitelné, a tak se pro symbolický zápis IP adres zavedla vhodná konvence, označovaná jako tečkovaná desítková notace (dotted decimal notation). Spočívá v tom, že 32 bitů IP adresy se rozdělí na čtyři části po osmi bitech (oktety), a každá z těchto částí se pak vyjádří jako celé desítkové číslo bez znaménka (s použitím tečky jako oddělovače jednotlivých částí). Například nepříliš mnemonický tvar 10000000 00000001 00000010 00000011 tak dostává přehlednější podobu: 128.1.2.3. Význam desítkové notace ilustruje též Tabulka 3.5.2.

	Obecný tvar IP adresy uvažujeme p.q.r.s			
	první oktet adresy	síťová část adresy	část adresy, představující relativní adresu uzlu v rámci sítě	maximální počet hostitelských počítačů
třída A	1-126	p	q.r.s	16777214
třída B	128-191	p.q	r.s	65534
třída C	192-223	p.q.r	S	254

Tabulka 3.5.2: Desítková tečkovaná notace a IP adresy

3.5.4. Porovnání IP adres a fyzických adres v počítačových sítích

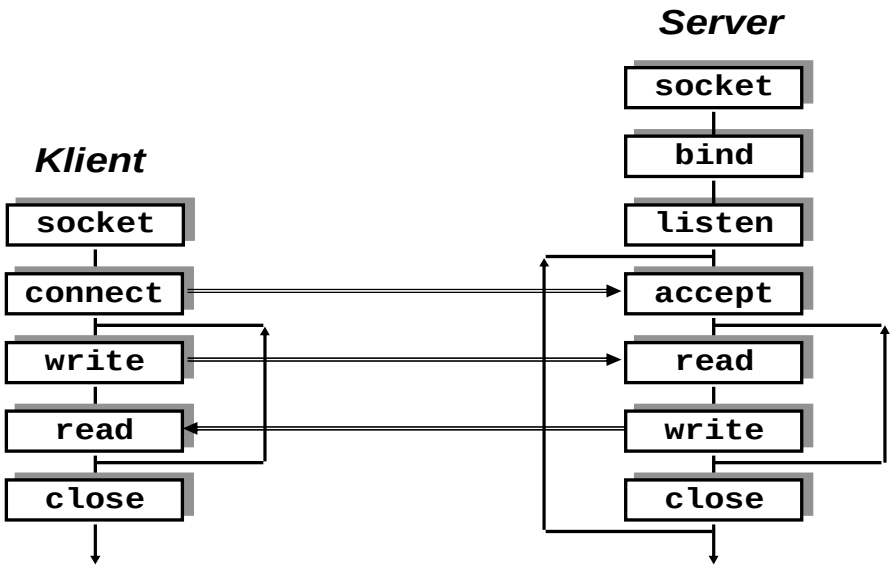
Lidé (aplikační vrstva)	Doménová adresa (např.: fel.cvut.cz)	- přidělována podle organizační struktury - snazší zapamatování
SW (síťová vrstva)	IP adresa (např.: 191.25.16.8)	- přidělována podle topologie - určuje jednoznačně síť a v jejím rámci počítač
HW (linková vrstva)	MAC (ethernetová) adresa (např.: 8:0:20:ea:6:1f)	- dána výrobcem - nerespektuje topologii

Tabulka 3.5.3: Konverze síťových adres

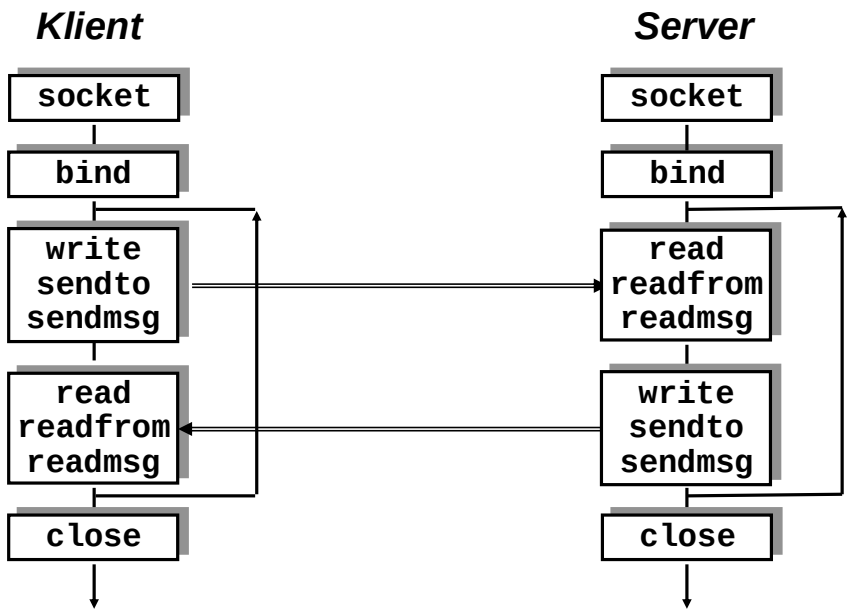
3.5.5. IPnG – výhled do budoucnosti

Způsob přidělování IP adres po celých třídách A, B a C časem začal být příliš neefektivní. Při prudkém rozvoji Internetu, a tím při nesmírné poptávce po IP adresách totiž začalo hrozit jejich vyčerpání. Radikálním a současně i definitivním řešením hrozícího nebezpečí mohl být pouze přechod na větší síťové adresy (s větším počtem bitů), ale to znamenalo úplně předělat celý protokol IP. Proto byl vyvinut nástupce současného protokolu IP (verze č. 4). Nový protokol IP, který již pracuje se 128 bitovými adresami, dostal jméno "IP nové generace" (IPnG, IP Next Generation), nebo je označován také jako IP verze 6 (IPv6, přičemž verze 5 neexistuje). Praktické nasazení tohoto protokolu je však zatím téměř nulové.

3.6. Přenos dat v sítích TCP/IP

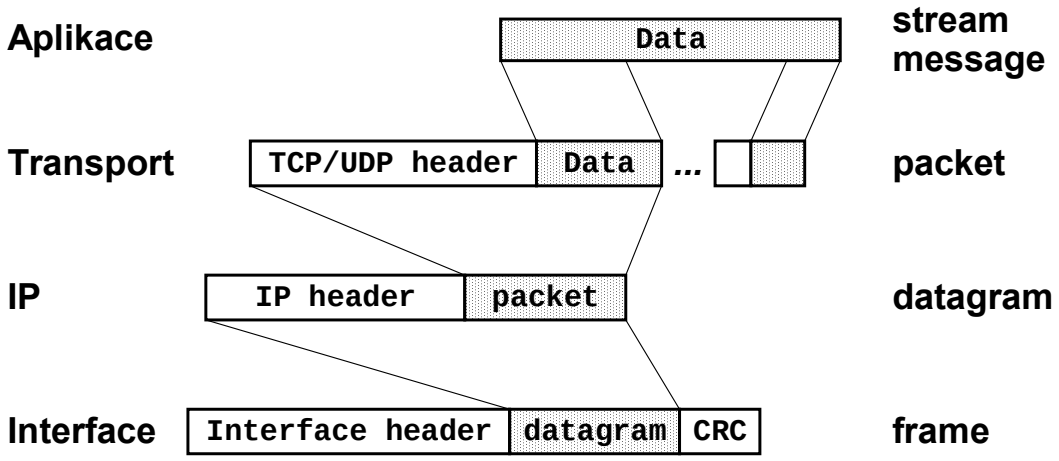


Obrázek 3.6.1: Standardní model TCP aplikace



Obrázek 3.6.2: Standardní model UDP aplikace

3.7. Struktura dat v TCP/IP



Obrázek 3.7.1: Struktura dat v TCP/IP rámcích

3.8. Implementace TCP/IP v prostředí Borland Builder

Ve vývojovém prostředí Borland Builder je TCP/IP socket spojení obstaráváno pomocí standardních objektů (komponent) *TServerSocket* a *TClientSocket*. Tyto objekty kompletně obstarávají a zabalují všechny aspekty pro TCP/IP spojení. Pomocí těchto komponent je možné naprogramovat síťové servery a klienty, kteří mohou číst data z jiných

aplikací. Užívat lze libovolně jakékoliv protokoly a porty v rámci protokolu TCP/IP.

Při uzavření spojení mezi serverem a klientem, jsou jednotlivé koncové body spojení ekvivalentní a z hlediska vlastností je nelze rozlišit. Kompletní socket spojení je plně popsáno adresami socketů na obou koncích spojení, jen takováto spojení mohou být realizována. Každé spojení musí mít své unikátní číslo (identifikaci), přesněji různá spojení mezi stejnými IP adresami musejí mít alespoň na jedné straně spojení různá čísla portu. Číslo portu se obvykle mění na straně klienta, pokud služba poskytovaná serverem zůstává stejná. O automatické přidělování čísla portu na straně klienta se stará systém při navazování spojení.

3.8.1. Spojení ze strany klienta

Pro realizaci spojeni na straně klienta se používá objekt `TClientSocket`. Nejprve je nutné definovat spojení, které chceme uskutečnit. To znamená zadat IP adresu serveru a port na jakém je patřičný server připojen. Poté se již pouze zavolá metoda objektu a ta naváže požadované spojení nebo vyvolá výjimku s popisem vzniklé chyby.

3.8.2. Spojení ze strany serveru

Na straně serveru je realizace spojení poněkud složitější. Samozřejmě musíme specifikovat port, kde server bude poskytovat spojení. Spojení může být dvou typů: „non-blocking connection“ a „blocking connection“. Tyto se od sebe liší způsobem jak probíhá čtení a zápis dat přes spojení.

- **non-blocking connection** – Při tomto typu spojení je čtení a psaní dat do socketu asynchronní. To má za výhodu, že při běhu programu nečeká provádění zbytku kódu na dokončení obsluhu spojení.
- **blocking connection** – Zde čeká aplikace na dokončení čtení nebo zápisu přes spojení. Při realizaci tohoto spojení je vhodné používat více vláknový („multi thread“) model aplikace.

Při realizaci obou typů koncových bodů spojení (klient a server) je po navázání spojení vlastní komunikace realizována přes objekty typu stream (tok dat, proud dat). Na obou stranách spojení jsou tedy naprosto identické toky dat, do kterých lze zapisovat nebo z nich číst (pokud obsahují čekající data). Jakmile jsou na jedné straně data do toku zapsána, druhá strana je informována, že přišla nová data a jsou připravena k přečtení. Pokud by na přenosové trase došlo k výpadku části nebo všech dat je vyvolána chybová událost. Zajišťování bezchybnosti přenosu a detekce chyb je obstaráváno protokolem TCP. Tak na obou stranách spojení je zaručena identifikace chyby přenosu.

4. SQL server

Zde uvádím jaké důvody vedly k rozhodnutí použít v systému databázový server MySQL (kapitola 4.1). Budou popsány vlastnosti a omezení tohoto serveru v kapitole 4.2. Principy ze kterých vychází manipulace s daty uvnitř MySQL (kapitola 4.3). A v kapitole 4.4 uvedu hodnocení ohledně stability serveru. Nakonec uvedu několik poznámek o použitém databázovém rozhraní ODBC (kapitola 4.5) a o jeho implementaci ve vývojovém nástroji Borland Builder C++ (kapitola 4.6). Informace o testech rychlosti jsou v příloze 9.2 Rychlost serveru MySQL, výsledky testů.

4.1. Proč MySQL, filosofie MySQL

Jako databázový server SQL, který je začleněn do systému pro sběr a zpracování dat je vybrán server MySQL [1] . Důvody, které k tomu vedly jsou, že tento server je poskytován zdarma, ke stažení na Internetu, pro používání v souladu s podmínkami GNU General Public License (GPL) (http://www.gnu.org/licenses/). Zároveň je ale poskytována i druhá licence, komerční. Vlastní program zůstává pro obě licence stejný, ale liší se služby, které zákazník s programem získává. Jde hlavně o další rozšíření a nástroje k programu a poskytování poradenských služeb („helpdesk“) od výrobce, firmy MySQL AB, a od dalších „third party“ společností. Dalšími důvody pro použití MySQL jsou jednoduchost instalace a práce s databázemi. Výkon serveru MySQL při zpracování dat předčí i některé komerční servery prodávané za velmi vysoké ceny.

Proč je MySQL zadarmo? Fakt, že server MySQL je zdarma, by mohl působit dojmem, že jde o ne příliš kvalitní systém. Ale skutečnost je jiná. Jde o marketingovou strategii firmy vyvíjející tento produkt. Firma se snaží o co největší rozšíření svého produktu. Peníze, které získává plynou hlavně z prodaných licenci, z poskytování poradenské a konzultační činnosti a z tréninkových kurzů pro produkty MySQL AB a související. Dalším zdrojem financí je i reklamní činnost, například reklamní plochy poskytované přímo na serveru www.mysql.com, který je hojně navštěvován odborníky v informačních technologiích, pracujícími s MySQL.

Kdy je tedy potřeba koupit komerční licenci produktu? Tyto omezení vyplývají z definic GNU GPL. Licence GPL je dodržována pokud program je spojen výhradně s produkty splňující GPL a všechny jeho zdrojové kódy všech jeho částí jsou distribuovány pod licencí GPL. Komerční licence je tedy potřeba hlavně v případech:

- pokud k výslednému produktu nejsou dodávány zdrojové kódy
- pokud není výsledný produkt v licenci GPL a spolupracuje pouze s produktem GPL (což je např. MySQL)

[1] Dále popisovaná verze serveru MySQL je 3.23.52

Je li tedy výsledný produkt, vytvořený v libovolném (třeba komerčním) programovacím nástroji (např. Borland C++ Builder), šířen v licenci GPL, není nutné zakoupení komerční licence MySQL. Je však nutné brát na zřetel smluvní podmínky sjednané s poskytovatelem programovacího nástroje, ve kterém byl výsledný program vytvořen.

Hlavní vytčené cíle, které se snaží MySQL databáze dodržovat jsou:

- nejlepší a nejvíce světově dodržovaná databáze
- přístupná a poskytnutelná pro všechny
- jednoduchá na používání
- neustále se vyvíjející
- uživatelsky příjemná
- neobsahovat chyby

V neposlední řade stojí za to se zmínit i o tom, že jedna z motivací k šíření databáze MySQL licencí GNU GPL je snaha zabraňovat nelegálnímu šíření kradeného softwaru. Cesta k tomuto vede právě poskytováním databázového systému MySQL zdarma.

4.2. Specifika a omezení MySQL

4.2.1. Rozdíly oproti ANSI SQL92

Databáze MySQL obsahuje některá rozšíření oproti norně SQL92. Může se tedy stát, že kód spustitelný na MySQL je nepřenosný na jinou databázi. V některých případech lze klíčová slova určená pouze pro MySQL označit pomocí definovaných symbolů („/*!“, blíže popsáno v [6]) a tak budou ostatními databázovými servery ignorovány.

např.: SELECT /! STRAIGHT_JOIN */ col_name FROM table1,table2 WHERE ...*

Další podstatnou věcí je, že některé funkce a chování předepisované normou SQL92 nejsou do MySQL implementovány. Mezi nejvýznamnější paří rozdíl, že pro sloupce typu VARCHAR jsou při ukládání automaticky odstraňovány „netisknutelné znaky“ (trailing spaces). Dále jsou zde některé rozdíly v přístupových právech a vyhodnocování logických výrazu. Pro podrobnější informace odkazuji na [7].

Významným vlastností MySQL při zpracování a manipulaci s daty je absence možnosti použití vnořených selektů (`SELECT * FROM table1 WHERE id IN (SELECT id FROM table2);`) V mnoha případech lze tuto limitaci obejít pomocí jiné vhodné konstrukce.

Na tomto databázovém serveru není rovněž implementováno vyvolávání události při nějaké konkrétní změně dat. Důvodem k tomuto je zachování rychlosti zpracování příkazu serveru.

Zatím chybí také možnost ukládání procedur na server. S touto vlastností se ale počítá do budoucnosti a je na seznamu věcí, které se tvůrci budou snažit k serveru přidělat.

Do tohoto seznamu zatím stále patří také možnost používání „foreign keys“. Zatím je implementováno, že procesor SQL příkazů tato používaná klíčová slova zná v příkazu CREATE TABLE, kde je nevyhodnocuje za chybu.

Ale ve skutečnosti tato klíčová slova nedělají nic, jsou zde pouze pro zachování kompatibility a přenositelnosti příkazů.

Poslední nepodporovanou konstrukcí jazyka SQL je vytváření náhledů na tabulky „view“. I s tímto se však počítá do budoucích verzí.

4.2.2. Transakce a MySQL

V současné verzi jsou serverem MySQL transakce podporovány pro všechny typy dat „transaction safe“. Více je o tomto problému uvedeno v uživatelském manuálu k MySQL [6] a dále, v kapitole 4.3 Základní principy v MySQL.

4.3. Základní principy v MySQL

Filosofie přístupu k databázím a bezpečnosti dat v nich, ze které se snaží vycházet tvůrci MySQL, je dodržovat paradigma datové integrity odvíjející se z používání atomických operací. Podle vypracované teorie a výsledků z praxe je zřejmé, že toto řešení poskytuje ekvivalentní či dokonce lepší datovou bezpečnost oproti vytvoření transakční databáze.

Atomické operace znamenají, že nemohou nastat takové situace, kdy zároveň za běhu jedné operace s daty by jiný uživatel změnil část dat nacházejících se jinde v databázi. V této filosofii také platí, že nikdy nedojde k automatickému „rollbacku“ dat, jak se může stát při používaní transakcí. Tímto způsobem je zajištěno, že nikdy nemůže dojít k situaci čtení neplatných či nekonzistentních dat („dirty read“) vinou databáze.

Nespornou výhodou používaní atomických operací je významné zvýšení výkonu oproti jiným řešením. Výhledem do budoucna však MySQL bude podporovat jak transakční paradigma, tak i atomické, a bude na volbě uživatele, co se mu více hodí do jeho aplikace. Zda používaní atomických operací a tím zajištění vysokého výkonu a rychlosti, nebo používání transakčních principu ale s daní snížení výkonu.

4.4. Stabilita MySQL

Stabilita MySQL byla testována po několik let ve firmě TCX Data Consult AB, kde byla vyvinuta a použita v jednom z jejich projektů. Zde fungovalo MySQL v jejich aplikacích bez problémů od roku 1996. Po uvolnění programu pro veřejnou sféru bylo, díky novým programátorským přístupům lidí mimo firmu, odhaleno několik chyb. Všechny chyby, které byly způsobeny chybami v kódu, byly odstraněny. Zůstalo jen několik neodstranitelných chyb vyplývajících z architektury systému. Pokud se objeví nějaké nové chyby, jsou co nejdříve odstraněny v nové verzi programu. Celkově byl systém testován mnoha typy testů a lze říci, že ve všech MySQL dopadlo dobře.

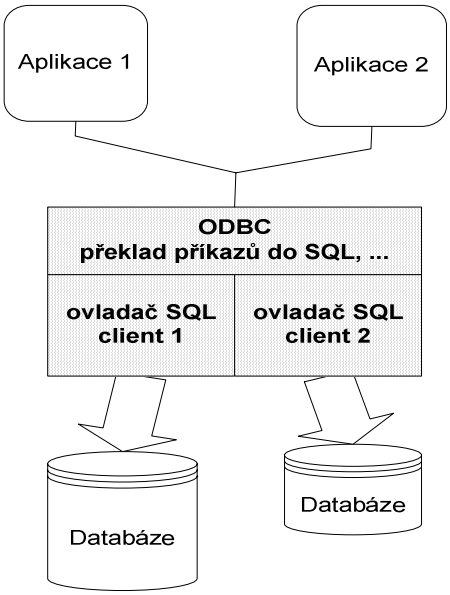
4.5. ODBC

ODBC (Open Database Connectivity) je rozhraní MS - Windows pro připojení se k databázi, definuje API (Aplication Programming Interface) pro přístup k databázím. Jeho záměrem je soustředit v sobě funkční oblasti společné pro mnoho databázových produktů. Je postaveno na specifikacích SAG a X/Open (na jejich raných pracích). Rozhraní ODBC je fyzicky realizováno tak, že pod společnými funkčními nástroji je sada samostatných ovladačů pro přístup ke konkrétním systémům. Aplikace pracuje s databází pomocí ODBC funkcí, které jsou uloženy v dynamické knihovně (DLL). Tyto funkce jsou určeny pro klienta databáze. Mezi DLL funkcemi a vlastní databází se nachází databázový ovladač, který převádí obecné požadavky na jakoukoli databázi do tvaru pro databázi konkrétní. Pro využívání ODBC je nutné mít ovladač dané databáze. Rozhranní ODBC se celkově skládá ze dvou hlavních částí. Je to část společná pro všechny datové zdroje a část určená ke komunikaci s konkrétním databázovým zdrojem. Část společná pro všechny datové zdroje se nazývá ODBC Driver Manager a komunikace s databázovým zdrojem zajišťuje ODBC ovladač (driver). Hlavním cílem ODBC je tedy umožnit psaní aplikačních programů nezávislých na databázi, s jejímiž daty budou pracovat a umožnit přístup do databází všem programům jednotným způsobem.

Podstatným rysem ODBC je fakt, že s hostitelským systémem lze komunikovat výhradně pomocí funkcí, které jsou v ODBC zabudovány. Omezením ODBC je to, že standardně neřeší navigační přístup k databázím. Může být realizován pouze v případě, že ho podporují ODBC ovladače jednotlivých systémů. Zároveň není uzpůsobeno pro zpracování dotazů z více datových zdrojů zároveň.

Důležitou vlastností ODBC je, že informace o tom, jak napsat ODBC ovladač, jsou volně k dispozici, takže ODBC ovladače mohou vytvořit nezávislé databázové firmy pro přístup ke svým datům. V zásadě lze ODBC ovladače rozdělit do dvou skupin. Ovladače pro souborové databáze musejí zajistit vykonávání veškerých příkazů ve vlastní režii. Naopak ovladače pro klient/server databáze pouze zajistí předání příkazu na server a převzetí výsledku ze serveru. Driver také musí zajistit zobrazení okna pro zadávání parametrů pro konfiguraci aliasu a případě okna pro přihlašování k databázi.

Pro manipulaci s daty používá ODBC standardní jazyk SQL. Veškerá manipulace s datovým zdrojem je ale realizována prostřednictvím sady funkcí ODBC API - např. pro přímé spuštění SQL příkazu se používá funkce SQLExecDirect. Tyto funkce pokrývají přihlašování k datovému zdroji, práci s SQL příkazy, získávání výsledků, práci s transakcemi, práci s kurzory, práci se systémovým katalogem, funkce pro ošetřování chyb a ještě další.



Obrázek 4.5.1: Funkce ODBC

Obrázek 4.5.1 zobrazuje více databázových systémů a více datových základen přístupných pomocí jednotného rozhraní ODBC. Kromě jednotného přístupu k ODBC, které tak zajišťuje přístup všech systémů podporující ODBC, nabízí tvůrcům aplikací již některé připravené funkce. Tyto funkce mohou být využity oběma systémy a nemusí být tak v obou samostatně řešeny (např. převod různých tvarů dotazu na SQL, síťový provoz hostitelského systému). ODBC nenabízí heterogenní dotazy a standardně neřeší navigační přístup. Dalším z nedostatků ODBC je, že neexistuje žádná standardní objektová nadstavba, která by zajišťovala pohodlnější manipulaci s daty. Bližší informace jsou například v [8].

4.6. Databáze, ODBC a Borland C++ Builder

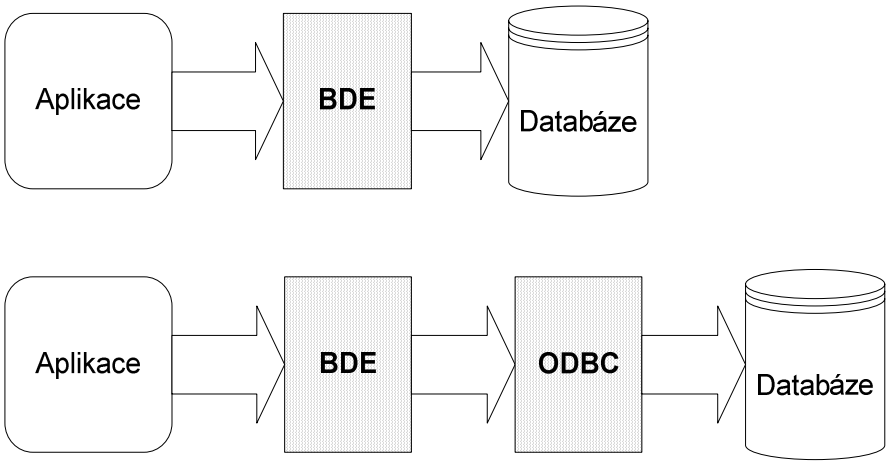
Borland C++ Builder poskytuje podporu pro relačně orientované databáze přes jeho rozhranní BDE (Borland Database Engine). Architektura BDE je založena na ovladačích, které musejí být konkrétně pro daný druh databáze. Ovladače mohou být pro databáze založené na zcela rozličných podstatách. Jde například o lokální soubory různých typů, o celou řadu SQL serverů a ovladače typu ODBC (Open DataBase Connectivity). Interface BDE poskytuje mnoho různých objektů (knihoven) pro připojování se k databázím a práci a výměnu dat. Jmenuji některé význačné znaky BDE:

- BDE v sobě slučuje několik možných přístupů k databázím a uživateli nabízí jednotné rozhraní.
- Přes BDE je možné jednoduše realizovat výměnu dat mezi různými druhy databází.

- Přes BDE je možné jednoduše realizovat dotazy zahrnující v sobě několik druhů databází. Lze například realizovat dotaz na data z tabulky dBase a Oracle současně.
- BDE nabízí výběr z několika dotazovacích jazyků – SQL, QBE.
- BDE poskytuje plnou podporu 32-bitových systémů zahrnující multi-thread architekturu, preemptivní multitasking, universal naming convention (UNC), dlouhá jména souborů, atd.

Objekty BDE jsou zabaleny do několika vizuálních i nevizuálních komponent. Užívání těchto komponent je jednoduché a uživatelsky přívětivé.

V BDE je rovněž zahrnuta podpora transakcí v databázích, avšak pouze pro databáze, které tuto architekturu přístupu nativně podporují a částečně pro ovladače ODBC.



Obrázek 4.6.1: Architektura přístupu k databázím přes BDE

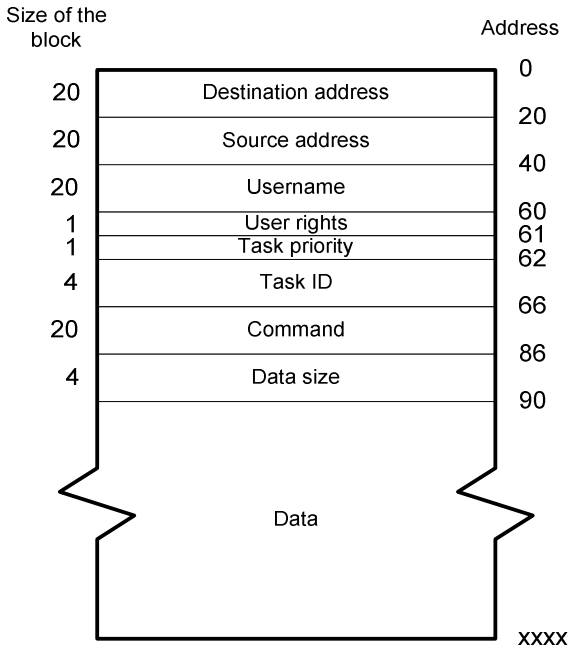
5. Implementace systému

V tomto oddíle bude postupně popsána implementace všech naprogramovaných částí systému s popisem jejich architektury, vlastností a funkcí, které poskytují přes jejich síťová rozhraní. Jde o celý balík programů obsahující server s názvem UniServer, k němu 19 klientů, zprostředkovávající rozličné funkce, a jedna komponenta pro Borland C++ Builder. Síť tvořící UniServer s klienty bude dále v textu nazývána UniNetwork.

Jako vývojový nástroj pro implementaci systému byl zvolen produkt Borland C++ Builder (verze 5.0). Jedním z důvodů pro tento nástroj je, že v tomto prostředí je jednoduchá tvorba grafického uživatelského prostředí (GUI). Dále je zde velmi zdařile řešeno používání komponent v programech a je dostupná jejich rozsáhlá knihovna. Tyto komponenty značně usnadňují a zrychlují vývoj celé aplikace. Dále je zde mohutná podpora vývoje aplikací typu ActiveX, které jsou použity v systému. Funkčnost a možnosti kódu jazyka C++, zde implementovaného, jsou zcela vyhovující a pro danou potřebu stejné jako v jiných vývojových nástrojích. Tyto vlastnosti dohromady umožňují programátorovi soustředit se na vlastní problémy a algoritmy v budovaném systému a nevěnovat velkou pozornost a hlavně spoustu času problémům, které s jeho prací přímo nesouvisí.

Všechny součásti programového balíku UniNetwork jsou kompatibilní se všemi verzemi operačního systému Windows s 32 bitovým jádrem.

5.1. Struktura paketů (úloh) přenášených v systému



Tabulka 5.1.1: Struktura paketů serveru

Paket (úloha) přenášený uvnitř systému má dvě hlavní části. Mohli bychom je pojmenovat část informativní a část datová. V první části jsou neseny informace sloužící k adresování a identifikaci daného paketu v systému. Ve druhé části to jsou předávaná data, která blíže specifikují či jinak souvisejí s úlohou (příkazem) daného typu, který je určen položkou „Command“.

V souladu s označením jak uvádí Tabulka 5.1.1 je popis významu položek datové struktury úlohy následující:

Destination address

Datový typ: charakter string, délka max 20

Určuje adresu (jméno) klienta pro kterého je paket určen. Na tuto adresu je paket směřován po průchodu serverem.

Source address

Datový typ: charakter string, délka max 20

Určuje adresu která daný paket vyslala, resp. na kterou má být doručena odpověď. Její obsah může vysílající klient libovolně měnit a tím může docílit toho, že odpověď na jeho požadavek dostane jiný klient. To je výhodné pro řízení předávání dat v systému. Jeden klient, který se sám neúčastní výměny dat, řídí tyto výměnu mezi jinými dvěma klienty (použito např. v klientu „DataPump“).

Username

Datový typ: charakter string, délka max 20

Identifikuje uživatele, který je přihlášen na odesílajícím klientovi. Její obsah uživatel sám nesmí měnit, po přihlášení již musí zůstat stejná. Případné pokusy o změnu server kontroluje a navrací zpět chybovou hlášku. Položka má význam kvůli řízení autorizace přístupu ke klientům systému.

User rights

Datový typ: byte

Specifikuje práva uživatele (0 - 255) přidělená administrátorem systému. Její obsah uživatel sám nesmí měnit, po přihlášení již musí zůstat stejná. Případné pokusy o změnu server kontroluje a navrací zpět chybovou hlášku. Položka má význam kvůli řízení autorizace přístupu ke klientům systému.

Task priority

Datový typ: byte

Určuje prioritu úlohy (0 - 255). Její hodnotu může vysílající klient nastavovat libovolně. Její hodnota nijak neovlivňuje přenos dat v serveru, pouze má informační charakter pro přijímajícího klienta.

Task ID

Datový typ: unsigned int

Vysílající klient může při vysílání požadavku specifikovat její hodnotu na libovolné číslo. Odpověď na požadavek je vracena se stejnou hodnotou. Umožňuje identifikovat odpověď na požadovaný dotaz. Je definováno několik hodnot mající speciální význam:

4294967281 - IDGivenByConstructor
4294967282 - IDForDefaultIncomeTask
4294967283 - IDForTesting
4294967284 - IDOfUnaskedTasks
4294967285 - IDOfTasksWithoutAnswer

Command

Datový typ: charakter string, délka max 20

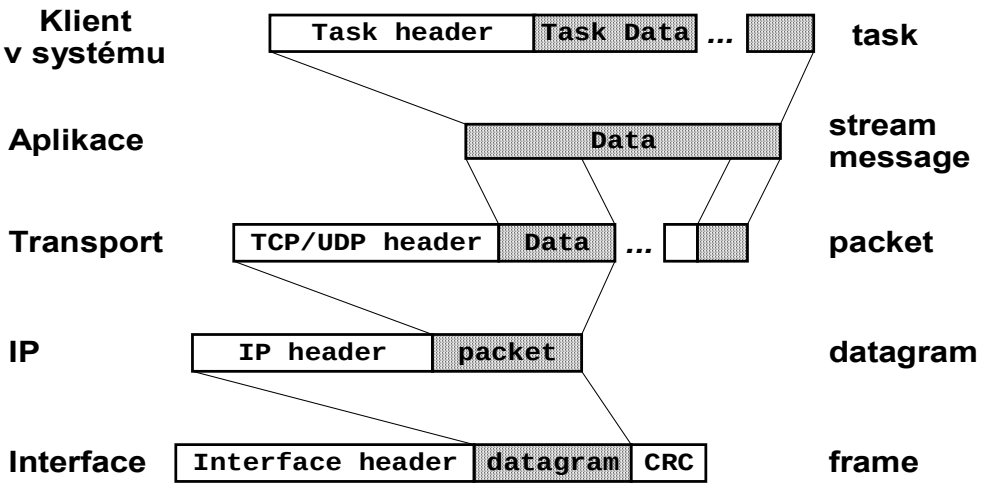
Určuje typ paketu a strukturu dat, která jsou nesena v datové části paketu. Informace slouží pro cílového klienta, který musí příkaz znát aby mohl správně přečíst a interpretovat data úlohy.

Data Size

Datový typ: unsigned int

Velikost dat obsažených v paketu. Je nastavována automaticky při přijmutí paketu nebo při zápisu do něj. Uživatel může její hodnotu pouze číst.

Používání takto definované paketové komunikace znamená obsazení protokolu aplikační vrstvy v modelu TCP/IP komunikace. Graficky to znázorňuje následující obrázek (porovnejte s modelem struktury dat v TCP/IP rámcích, kterou zobrazuje Obrázek 3.7.1)



Obrázek 5.1.1: Rozšíření modelu TCP/IP

Na obrázku jsou vidět dvě části struktury úlohy. Je to část nesoucí informace o typu, velikosti, cílové adrese a další specifikace úlohy (Task header) a část obsahující vlastní data úlohy (Task data), Jde o logické dělení úlohy v souladu s popisem uvedeným v kapitole 5.1.

5.2. Výměna proměnných v síti UniServeru (UniNetwork)

Pro výměnu proměnných po síti UniNetwork slouží dva typy paketů. Je to paket s příkazem „GETDATA“ a příkazem „PUTDATA“. Abychom byli zcela přesní s výměnou dat souvisí ještě jeden paket a to „DATASAVED“. Cyklus výměny dat probíhá takto. Pokud si některý klient chce od jiného klienta vyžádat nějakou specifickou proměnnou (více v kapitole 5.2.1) zašle mu vyplněný paket „GETDATA“ (viz seznam podporovaných příkazu jednotlivých klientů). Na tento příkaz dotazovaný klient odešle zpět paket „PUTDATA“ s naměřenou hodnotou a časovou známkou.

Druhá situace je, že klient změří hodnotu svojí proměnné a chce ji zaslat nějakému jinému klientovi. Pak pošle paket „PUTDATA“. Jako odpověď, že paket přišel v pořádku data jsou na cílovém klientu v pořádku uložena obdrží nazpět paket „DATASAVED“

Tyto tři typy paketů (úkolů) jsou základními prostředky pro výměnu hodnot proměnných mezi klienty v celém systému. Proto musí být a jsou podporovány všemi klienty v UniNetwork, kteří se účastní výměny hodnot proměnných. Mimo tyto pakety může každý klient podporovat i další typy paketů, ve kterých je schopen předávat či přijímat data. Použití těchto rozšířených funkcí výměny dat již pak závisí na schopnostech a vzájemném nakonfigurování obou klientů mezi kterými výměna probíhá.

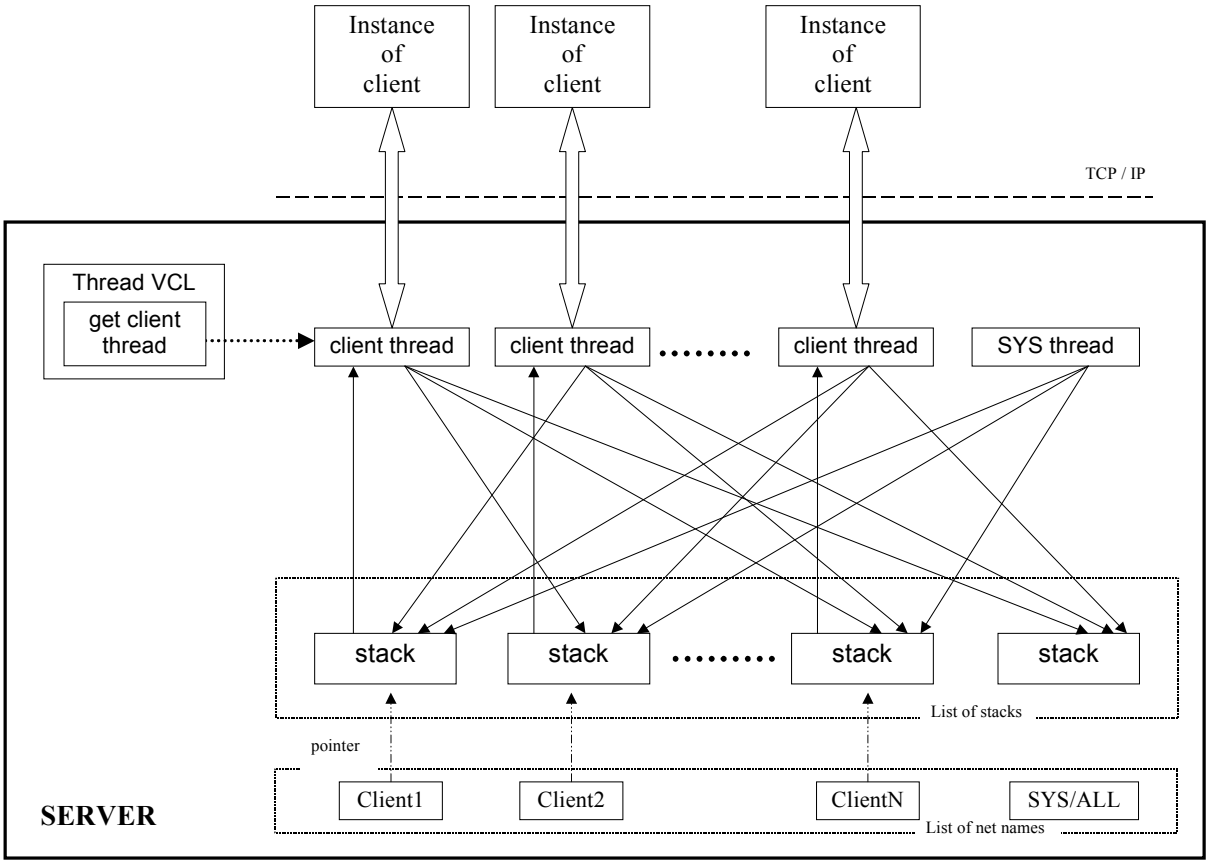
5.2.1. Konvence pojmenovávání proměnných

V celé síti UniNetwork je dodržována jednotná konvence pojmenovávání proměnných. Jméno každé proměnné se skládá ze dvou částí. První část slouží pro informaci od kterého klienta daná proměnná pochází. Druhá část je již vlastní jméno proměnné. Tyto dvě části jsou od sebe odděleny znakem obrácené lomítko – „\“.

Uvedeme si zde příklad. Předpokládejme, že do systému je připojený klient TestSignals, který je určený pro testovací a simulační účely. Na něm je nadefinována proměnná „Triangle1“. Dále je v systému připojen klient DataPump, sloužící pro stimulovaný přenos hodnot proměnných. Dále máme na server napojen klient „UniClient“, který obsahuje vizualizační rozhraní. Pak můžeme v nastavení DataPump zadat jako zdrojovou proměnnou „TestSignals:0\Triangle1“, což značí proměnnou jménem „Trinagle1“ fyzicky umístěnou na klientovi se jménem „TestSignals:0“. A jako cílovou proměnou zadat „UniClient:0\PrBar2“. Tak je systém nakonfigurován na zobrazování proměnné Triangle1 na panelu Demos na klientovi UniClient.

5.3. Server „ UniServer “

5.3.1. Architektura serveru a tok dat



Obrázek 5.3.1: Schéma architektury UniServeru

Podle logického dělení by se dal UniServer rozdělit na dvě základní části: část zajištující připojování a komunikaci s klienty po TCP/IP (součástí hlavního threadu VCL) a část obstarávající přihlašování klientů, správu připojení a přenášení dat (funkce threadu SYS).

Z obrázku je na první pohled zřejmé, že jde o vícevláknovou (multi-threadovou) architekturu serveru. Tato architektura je servery podobných typů používána ve většině případů, hlavně pro své příznivé vlastnosti. Pro nás důležitou vlastností je, že provádění komunikace s jedním klientem nijak neomezuje dokonce ani neovlivňuje (kromě ztráty části strojového času) komunikace ostatní. To je výhodné pokud spojení s nějakým klientem je velmi pomalé a při použití jednovláknové (single-threadové) architektury by na něj museli všichni ostatní čekat. Rovněž pokud nastane na některém spojení s klientem jakákoliv chybová situace (např. timeout), není tím zbytek systému nijak poznamenán. Dojde pouze k ukončení problémového spojení.

5.3.1.1. Thread VCL

Jde o hlavní thread, který obsahuje každá aplikace vytvořená v C++ Builderu, která používá knihovny VCL (Visual Component Library). Tento thread má v UniServeru několik úkolů. Jedním je samozřejmě zprostředkování interakce s uživatelem – administrátorem serveru. Dále se thread stará o zobrazování událostí vzniknuvších v systému.

Úloha threadu, která se týká komunikačního systému je reagovat na požadavky server socketu TCPIP o připojení nového klienta a vytvořit pro toto připojení vlastní thread a vlastní zásobníky - fronty na příchozí a odcházející úlohy.

5.3.1.2. Thread SYS

Thread serveru nazvaný SYS má za úkol obstarávat všechny události spojené s komunikací a bezpečností serveru.

Pokud se k serveru snaží připojit nový klient jsou pro něj vytvořeny všechny nezbytné systémové prostředky (viz kapitola 5.3.1.1) a očekává se příjem přihlašovacích informací od klienta. Ten musí zaslat úlohu typu *USERLOGIN* (viz níže v kap. 5.3.5). Na něj systémový thread SYS reaguje buď zasláním potvrzení (*USERLOGED*) o připojení do systému, nebo chybovou zprávou – úloha typu *ERRORMESSAGE* v případě neposkytnutí autorizace přístupu.

Reakce na všechny příkazy, které server podporuje obstarává právě thread. Další činností tohoto threadu je kontrolovat zda všichni připojení klienti jsou v připraveném stavu i když se zrovna aktivně neúčastní komunikace. To se děje prostřednictvím zasílaní kontrolních paketů *PING* v případě, že klient už 20 sekund neposlal žádný paket do systému. Pokud na tuto výzvu neodpoví je zobrazeno chybové hlášení a pokud odpověď nedojde do dalších 20 sec je spojení s klientem ukončeno.

Pokud cílová adresa úlohy je jiná než SYS a shoduje se jménem nějakého klienta v systému, je úloha zařazena do fronty úloh patřící příslušnému threadu a co nejdříve odeslán cílovému klientovi.

5.3.2. Nastavení serveru

V hlavním okně serveru se nalézá několik konfiguračních prvků. Nejvýznamnější je nastavení portu, na kterém server poskytuje připojení do systému. Standardně je číslo portu nastaveno na 8756. Na tomto portu již není podle doporučení TCP/IP žádná ze standardních služeb TCP/IP a je tudíž volný pro uživatelské služby (více kapitola 3). Jen bych ještě zopakoval, že na té samé IP adrese na jednom portu může být pouze jeden server. Číslo portu může uživatel libovolně specifikovat, avšak server musí být ve stavu Off Line (tlačítko vpravo dole).

Další nastavení na serveru je zaškrtávací políčko „Show debug information“. Při zaškrtnutí tohoto políčka se v okně hlášení o stavu serveru objevují všechny informace o přenesených tásících včetně dodatečných

informací o původu, velikosti, určení úlohy. Zapnutí této volby značně snižuje rychlost přenosu dat. K této volbě patří i tlačítko „Show messagees“, které otevře větší okno se systémovými hlášeními, které poskytuje lepší přehled v systémových hlášeních.

Volba „Place to icon tray “ určuje, zda bude server zobrazen jako normální programy na liště, nebo jestli bude zobrazen v okně ikon a služeb systému Windows.

5.3.3. Navázání komunikace, přihlášení a odhlášení

Po otevření socket connection mezi klientem a serverem klient zašle úlohu *USERLOGIN* na adresu „SYS“ s vyplněnými přihlašovacími údaji. Přístup je mu buď povolen *USERLOGED* nebo zakázán *ERRORMESSAGE* a je specifikován důvod proč tomu tak je. Po úspěšném přihlášení nebo po odhlášení (úloha *LOGOUT*) klienta je všem klientům připojeným do sítě rozeslána úloha *GETCLIENTLIST* nesoucí informaci o aktuálně připojených klientech.

Jako dvě speciální a rezervované adresy v systému jsou „SYS“ a „ALL“. Jak již jejich název napovídá, jedna přísluší systémovému threadu. Pokud pošle klient úlohu na adresu ALL, úloha bude doručena všem klientům v systému.

5.3.4. Kontrola komunikace, nastavení uživatelů

Administrátor serveru má možnost ovládat připojení klientů pomocí několika tlačítek umístěných v hlavním okně. Jejich názvy jsou natolik intuitivní, že je zde nebudu dále rozebírat (Obrázek 9.3.1: Hlavní okno UniServeru).

Nastavení uživatelských jmen a hesel uživatelů systému jsou uloženy v souboru „*users.txt*“. Formát souboru se shoduje se standardním formátem ini-souboru systému Windows. Tj.:

Username1=Password1
Username2=Password2
Username3=Password3

Načtení hesel do vnitřní cache serveru je provedeno při startu serveru. Pokud dojde ke změně v tomto konfiguračním souboru, je nutné tuto změnu ohlásit serveru stiskem tlačítka „Reload user config“.

5.3.5. Seznam příkazů podporovaných UniServerem

USERLOGIN

Popis: Tímto příkazem se klienti přihlašují do systému. Pro úspěšné přihlášení do systému musí být položky úlohy vyplněny následovně: Source Adres – požadovaná adresa v systému, Destination Address – nastavit na SYS, Requested rights – nastavit na 250. String v datové části musí mít formát USERNAME=PASSWORD
Data příkazu: AnsiString LogData
Význam dat: LogData – informace o uživateli a heslu, viz výše
Odpověď: USERLOGED

USERLOGED

Popis: Příkaz je zasílán serverem přihlašujícímu se klientovi v případě, že přihlášení do systému proběhne v pořádku.
Data příkazu: bool LogData
Význam dat: LogData – vždy true
Odpověď: -

ERRORMESSAGE

Popis: Příkaz je zasílán serverem přihlašujícímu se klientovi v případě, že přihlášení do systému neproběhne v pořádku, nebo dojde k jiné chybě při přenosu dat.
Data příkazu: AnsiString Text
Význam dat: Text – text specifikující danou chybu (např. Client doesn't exists, …)
Odpověď: -

INFOMESSAGE

Popis: Příkaz je zasílán serverem klientovi v případě nějaké události při přenosu dat. Příkaz má pouze informační charakter.
Data příkazu: AnsiString Text
Význam dat: Text – text s informaci pro klienta
Odpověď: -

LOGOUT

Popis: Příkaz zasílá klient, když se chce odhlásit ze systému.
Data příkazu: -
Význam dat: -
Odpověď: -

PING

Popis: Slouží pro test komunikace.
Data příkazu: -
Význam dat: -

Odpověď: -

PONG

Popis: Slouží pro test komunikace.
Data příkazu: -
Význam dat: -
Odpověď: -

GETCLIENTLIST

Popis: Příkazem žádá klient o zaslání seznamu aktuálně připojených klientů k serveru.
Data příkazu: -
Význam dat: -
Odpověď: GETCLIENTLIST
Data odpovědi: AnsiString List
Význam dat: List – seznam všech klientů přihlášených do systému. Jednotlivá jména jsou oddělena kontrolními znaky CR/LF

SUSPENDCLIENT

Popis: Příkazem je žádán server o pozastavení threadu patřící specifikovanému klientovi. Odpovědí na tento příkaz muže být i *ERRORMESSAGE* s textem „Client doesn't exists or is already suspended!“. Příkazy pro manipulaci s připojením ostatních klientů může zasílat jakýkoliv klient. Uživatelská práva zde nejsou kontrolována.
Data příkazu: AnsiString ClName
Význam dat: ClName – jméno klienta pro pozastavení
Odpověď: SUSPENDCLIENT
Data odpovědi: bool Succ
Význam dat: Succ – vždy true

RESUMECLIENT

Popis: Příkazem je žádán server o obnovení činnosti threadu patřící specifikovanému klientovi. Odpovědí na tento příkaz může být i *ERRORMESSAGE* s textem „Client doesn't exists or is already resumed!“
Data příkazu: AnsiString ClName
Význam dat: ClName – jméno klienta pro obnovení činnosti
Odpověď: RESUMECLIENT
Data odpovědi: bool Succ
Význam dat: Succ – vždy true

RESUMEALLCLIENT

Popis: Příkazem je žádán server o obnovení činnosti všech threadů.
Data příkazu: -
Význam dat: -

Odpověď: RESUMEALLCLIENT
Data odpovědi: bool Succ
Význam dat: Succ – vždy true

CLOSECLIENT

Popis: Příkazem je žádán server o odpojení specifikovaného klienta od systému. Odpovědí na tento příkaz může být i *ERRORMESSAGE* s textem „Client doesn't exists!“
Data příkazu: AnsiString ClName
Význam dat: ClName – jméno klienta pro odpojení
Odpověď: CLOSECLIENT
Data odpovědi: bool Succ
Význam dat: Succ – vždy true

CLOSEALLCONNECTION

Popis: Příkazem je žádán server o odpojení všech klientů. Po obdržení příkazu server pošle všem klientům *INFOMESSAGE* s textem: „Server shut down in 10 sec - command from user ….“
Data příkazu: -
Význam dat: -
Odpověď: CLOSEALLCONNECTION
Data odpovědi: AnsiString Text
Význam dat: Text – text: ” Server will close all connections in 10 seconds!!!”

5.4. Klient Builder komponenta, ActiveX klient

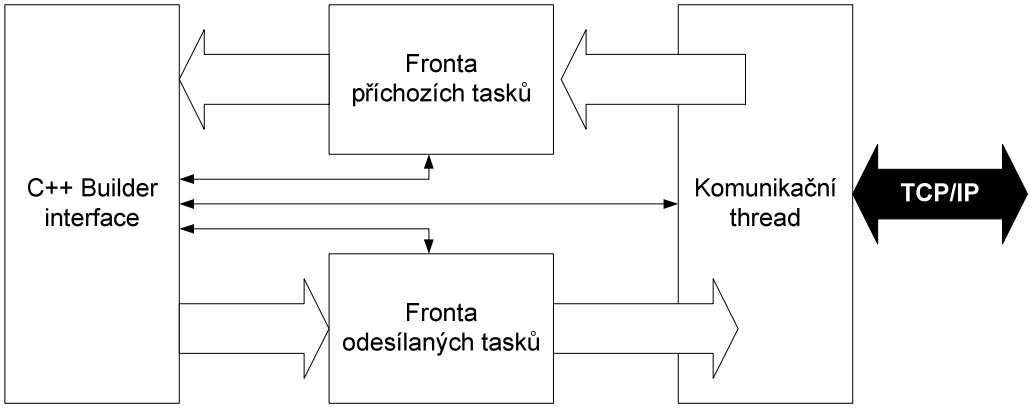
Tato komponenta je základním klientem systému. Zprostředkovává interface mezi připojením do sítě UniServeru a prostřením Borland C++ Builder. Je použita jako základ pro všechny další součásti (klienty) systému. Proto zde popíšeme podrobně nastavení a práci s touto komponentou a dále již připojení do sítě UniServeru nebudeme popisovat.

Předávání dat v komponentě probíhá pomocí popsané struktury úlohy. V objektu komponenty je implementována celá řada funkcí pro práci s těmito úlohami, pro přihlašování, přijímání úloh a další.

Jediný příkaz, který komponenta sama zpracovává a odpovídá na něj je příkaz PING, na který odpovídá příkazem PONG. Oba jsou s prázdnou datovou částí.

Přesné definice jednotlivých datových typů, proměnných a parametrů metod nebudu uvádět. Vše lze snadno zjistit ze souborů deklarace a implementace komponenty se jmény „UniConnection.h“ a „UniConnection.cpp“. Zde uvedu jen popis funkce metod a význam proměnných.

5.4.1. Architektura



Obrázek 5.4.1: Architektura komponenty univerzálního klienta

Komunikační thread kontroluje v cyklu jestli fronta úloh k odeslání obsahuje nějaká data, pokud ano, odešle je. Dále zkontroluje jestli nečekají nějaká data, která přišla přes TCP/IP. Pokud ano, přečte je a zařadí do fronty příchozích zpráv.

V dalším cyklu probíhá kontrola zda jsou ve frontě příchozích úloh data připravená ke čtení, pokud ano, je generována událost „OnWaitingResp“. Dále je již na aplikaci, která komponentu používá, aby si přečetla a zpracovala data.

Neustále se také kontroluje zda je komunikace v pořádku a schopná přenášet data. Pokud dojde k nějaké, chybě je generována událost „OnCommError“ a seznam chybových hlášení si je možno přečíst v proměnné „ErrorList“.

5.4.2. Datové položky komponenty

SrcUser: u příchozí úlohy určuje username odesílajícího uživatele. Pro odesílané úlohy nemá význam, je pouze ke čtení

Data: aktuální pointer na data v úloze

Task: aktuální pointer na úlohu

Host: při navazování komunikace určuje IP adresu počítače, kde běží UniServer

HostName: při navazování komunikace určuje jméno počítače, kde běží UniServer

NetName: při navazování komunikace určuje o jaké síťové jméno klient žádá.

UserName: při navazování komunikace určuje uživatelské jméno pro přihlášení

Password: při navazování komunikace určuje heslo uživatele pro přihlášení

Port: specifikuje port, na kterém probíhá komunikace

Requested rights: při navazování komunikace určuje jaká uživatelská práva v síti jsou žádány pro klienta

SettingsFormEnabled: určuje, zda je povoleno uživateli vyvolat formulář s nastavením pomocí stisku pravého tlačítka myši
NrOfWaitingIncomTask: počet příchozích úloh čekajících ve frontě
NrOfWaitingOutTask: počet odcházejících úloh čekajících ve frontě
CommInProgress: určuje zda připojení k serveru se otevřeno
ClientsList: seznam klientů připojených na server získaný při přihlášení, nebo po posledním voláním metody „RefreshAddressList“
ErrorPresent: je true pokud v seznamu chyb je nějaké chybové hlášení
ErrorList: seznam chybových hlášení od posledního vymazání seznamu. Jednotlivé položky jsou odděleny CF/LF
SrcAddress: adresa, ze které je úloha posílána
DesAddress: adresa, na kterou je úloha posílána
Command: příkaz pro klienta, identifikace typu úlohy
TaskID: identifikační číslo úlohy
Priority: priorita přiřazená úloze
DataSize: velikost datového pole úlohy
ActualTime: aktuální čas ve formátu používaném v systému pro výměnu dat mezi klienty

5.4.3. Metody komponenty

TimeStringToMsec: převod formátu času ze stringu na milisekundy
MsecToTimeString: a obráceně
StringToVariantArray: převod textu s položkami oddělenými CF/LF na jednorozměrné pole stringů, jde o pole typu variant
Connect: povel pro připojení k serveru
Disonnect: povel pro odpojení od serveru
RefreshAddressList: povel pro znovunačtení seznamu klientů
ShowSettingsForm: zobrazí formulář s nastavením

Metody pro práci s daty v táscích:

WriteDat: vymaže všechna data v datové části úlohy a zapíše do něj předaná data podle jejich typu. Všechny modifikace příkazu jsou: WriteDatBool, WriteDatByte, WriteDatLong, WriteDatSingle, WriteDatDouble, WriteDatString, WriteData (zde jsou z důvodu kompatibility s ActiveX rozhraním data předávána jako jednorozměrné pole typu variant s prvky byte)
AddDat: nevymaže data z úlohy a přidá na jejích konec další data. Všechny modifikace příkazu jsou: AddDatBool, AddDatByte, AddDatLong, AddDatSingle, AddDatDouble, AddDatString, AddData
ReadDat: čte data z místa datové části odkud bylo naposledy čteno. Po dokončení posune pointer ukazující aktuální pozici v datech. Všechny modifikace příkazu

	jsou: ReadDatBool, ReadDatByte, ReadDatLong, ReadDatSingle, ReadDatDouble, ReadDatString, ReadDatStringToVariantArray, ReadData, ReadDataAll.
ReadMatRMx:	pokud byla do úlohy zapsána data pomocí funkce WriteMatMX, nebo funkce AddMatMX, vrátí reálnou část matice a imaginární uloží do vnitřní paměti.
ReadMatIMx:	přečte imaginární část matice z vnitřní paměti, uloženou po posledním volání ReadDatRMx
ReadSQLInfo:	pokud je úloha odpovědí na příkaz EXESQLDAT, přečte informaci o počtu sloupců a řádků odpovědi a.o jejich typu
ReadSQLReply:	pokud je úloha odpovědí na příkaz EXESQLDAT, přečte tabulku z úlohy
ReadTimeVar:	pokud je úloha odpovědí na příkaz GETTIMEVAR, přečte hodnoty proměnné z úlohy
WriteMatIMx:	smaže úlohu a zapíše do ní matici, vhodné pro použití v programu Matlab
AddMatIMx:	to samé, ale nesmaže nejdřív původní data
SwitchAddress:	prohodí SrcAddress a DesAddress v hlavičce úlohy
SendTask:	odešle úlohu k serveru s nastaveným ID v úloze
SendTaskAutoID:	odešle úlohu k serveru s ID generovaným počitadlem odeslaných úloh
RecieveTask:	načte úlohu z fronty čekajících úloh do vnitřních proměnných, jen tak lze přistupovat k datům v úloze
RecieveTaskNr:	to samé, ale pokouší se načíst úlohu se zadaným ID. Čeká na ni po nastavenou dobu. Pokud se to nepovede načte se prázdná úloha
RecieveTaskCmd:	obdoba předchozího ale čeká se na úlohu se specifikovaným políčkem Command
ResetTask:	smaže všechna data v aktuálně zpracovávané úloze
ResetReadPointer:	nastaví pointer rozdělující data již z úlohy přečtená a data ještě nepřečtená na počátek
ClearTaskData:	smaže celou datovou část v aktuální úloze
SynchroAcquire:	při používání komponenty ve více threadech najednou slouží k zamknutí komponenty pro volající thread
SynchroRelease:	odemyká komponentu po předchozím volání SynchroAcqire

Příklady použití metod jsou uvedeny v přílohách 9.3.2.1, 9.3.2.2, 9.3.19.1.

5.4.4. Konfigurace komponenty, konfigurace v klientech

Formuláře pro konfiguraci připojení k UniServeru ukazuje Obrázek 9.3.2 a Obrázek 9.3.3 v příloze. Na prvním je formulář, který obsahuje vlastní

komponenta a na druhém je formulář (část formuláře) používaný ve všech klientech v systému. Formulář komponenty lze vyvolat kliknutím pravého tlačítka myši na komponentu. Položky a editační pole se v podstatě shodují. Zde popisuji, co jednotlivé položky znamenají a v dalších klientech se již otázkou připojení k UniServeru nebudu zabírat.

U většiny položek je naprosto zřejmé co se do nich vyplňuje. Jediná komplikovanější věc je vyplnění položky „Net name“. Ta se skládá ze dvou částí. První určuje jméno klienta a o druhé by se dalo říci, že jde o pořadové číslo klienta s tímto jménem. Při stisku tlačítka „Connect“ se klient pokusí připojit pod jménem složeným z těchto dvou částí oddělených dvojtečkou. Pokud se připojení nezdaří, protože v systému už je klient tohoto jména přihlášen, automaticky se inkrementuje číslo za dvojtečkou a pokus se opakuje. Takto se děje dokud pokus o přihlášení není úspěšný. Celkové jméno, pod kterým je klient v síti skutečně přihlášen, lze po úspěšném přihlášení přečíst v položce „Net name“. Jméno, tedy i adresa v síti, může vypadat např. „UniClient:0“

Ještě vysvětlím význam velké černé (bílé na formuláři který je součástí komponenty) tečky na připojovacím formuláři. Po úspěšném připojení se místo tečky objeví hodiny, jejichž chod vizuálně informuje o funkčním připojení k UniServeru.

Příklad konkrétního použití této komponenty v C++ Builder a této komponenty jako prvku ActiveX v jazyce Visual Basic pro řešení daného úkolu je uvedeno v přílohách 9.3.2.1 a 9.3.2.2.

5.5. SQL Bridge

SQL bridge je klient systému, který zprostředkovává jiným klientům, napojeným do systému, přístup k databázím. Je to vlastně propojení libovolného klienta s databází, odtud název Bridge. Přístup je realizován formou příkazů pro databázi, které jiný klient zasílá na adresu SQL Bridge. Tyto příkazy SQL Bridge provede a výsledky zašle zpět klientovi, který příkaz poslal. Seznam příkazů podporovaných SQL Bridge je uveden v příloze 9.3.3.1. V příloze je místěn i obrázek hlavního okna s konfigurací.

5.5.1. Databáze, návrh a struktura

Při návrhu databáze byly uvažovány požadavky:

- co nejrychlejší přístup
- úspora místa na disku
- jednoduchá a přehledná struktura dat
- možnost definování vztahu mezi jednotlivými proměnnými
- snadná identifikace proměnných v systému

ty vedly na jasné definování struktury databáze:

- ukládat se budou proměnné jen několika typů
- každý typ proměnné bude mít svoji vlastní tabulku
- identifikační informace o proměnných a o jejich vzájemném vztahu budou ve zvláštní tabulce
- hodnota každé proměnné u sebe bude mít časový údaj

Proto jsem vytvořil následující strukturu tabulek. Typy proměnných, které se v systému mohou vyskytovat jsou tři. Číselné hodnoty jsou rozděleny na dva typy, celočíselné (v databázi označované typem „INT“) a čísla s plovoucí řádovou čárkou (označované „FLOAT“). Třetím typem je textová proměnná (značená „STR“). Všechny ostatní datové typy musí být na tyto základní konvertovány. Konverze je ponechána na programu, který chce hodnotu do databáze uložit. Nejjednodušší a ve většině případů proveditelná je konverze na textový typ. Tato konverze většinou však není optimální ani z časového hlediska a ani pro manipulaci s hodnotou proměnné. Výsledná struktura tabulek v databázi je tedy následující:

Název položky
ID
Name
Type
ParetID
Info

Tabulka definice typu a struktury proměnných

Název položky
ID
ReadTime
ReadTimeMSec
Value

Tabulka pro ukládání proměnných

Tabulka 5.5.1: Tabulky databáze používané v SQL Bridge

Skript na definici struktury tabulek používaných programem SQL Bridge naleznete v příloze Tabulka 9.3.1.

Význam položek v tabulce definic je následující. Políčko ID značí unikátní identifikaci proměnné se jménem v políčku „Name“, která je datového typu „Type“. Textový popis významu proměnné je uložen v „Info“. Políčko „ParentID“ udává, která proměnná je dané položce nadřazena ve stromu hierarchie proměnných.

Položky v tabulce uložení proměnných jsou: „ID“ je identifikace položky (z tabulky definic), „ReadTime“ je čas změření hodnoty ve formátu „day.month.year hour:minute:seconds“. Políčko „ReadTimeMSec“ je pak upřesnění času na tisíciny sekundy. A políčko „Value“ obsahuje změřenou hodnotu proměnné. V tabulce je nastaven unikátní primární klíč ze sloupců ReadTime, ReadTimeMSec a ID, není tedy možné uložit dvě hodnoty téže proměnné se stejným časovým údajem, na tento pokus reaguje SQL Bridge vrácením chybové hlášky „ERRORMESSAGE“.

Tabulky jsou v systému pojmenované takto: tabulka s definicí proměnných a jejich struktury se jmenuje „vardef“, tabulky pro uložení hodnot proměnných jsou podle typu „varfloat“, „varint“ a „varstr“.

Soubory zkušební databáze jsou přiloženy na CD v adresáři Examples. Adresář „MyDbName“ stačí nakopírovat do adresáře „mysql\data“ a nakonfigurovat ODBC alias se jménem „MojeMySQL“ zastupující tuto databázi. Jména databáze mohou být libovolná, ale je nutno správně nakonfigurovat ODBC manager a SQL Bridge.

5.5.1.1. Paměťové nároky

Důležitým faktorem ovlivňujícím velikost databáze vztaženou na počet uložených položek bylo rozhodnutí ukládat každý datový typ do zvláštní tabulky. To má totiž za následek, že každá datová položka zabírá přesně tolik místa na disku, kolik je to nezbytně nutné. Pokud by byla nadefinována pouze jedna tabulka docházelo by k plýtvání místem v případě, že by se ukládala položka kratšího datového typu než jaká je nadefinovaná velikost sloupce tabulky (odpovídající nejdelšímu datovému typu).

Příklad výpočtu zabíraného místa na disku tabulkou databáze:

A/ Typ „INT“ - Počet uložených záznamů: 88104
Velikost místa obsazená na disku: 3682 KB
⇒ Uložení jedné položka zabírá: 41,79 byte
⇒ Odhad: ukládáme jednou za 100 ms
obsazené místo za den: 36 MB
obsazené místo za měsíc: 1,1 GB

B/ Typ „FLOAT“ - Počet uložených záznamů: 25750
Velikost místa obsazená na disku: 1318 KB
⇒ Uložení jedné položka zabírá: 51,28 byte
⇒ Odhad: ukládáme jednou za 100 ms
obsazené místo za den: 44 MB
obsazené místo za měsíc: 1,3 GB

Pro proměnné typu „STR“ nemá smysl provádět výpočet, protože velikost obsazená na disku je závislá na délce ukládaného textového řetězce.

5.5.1.2. Funkce, přístup k databázi

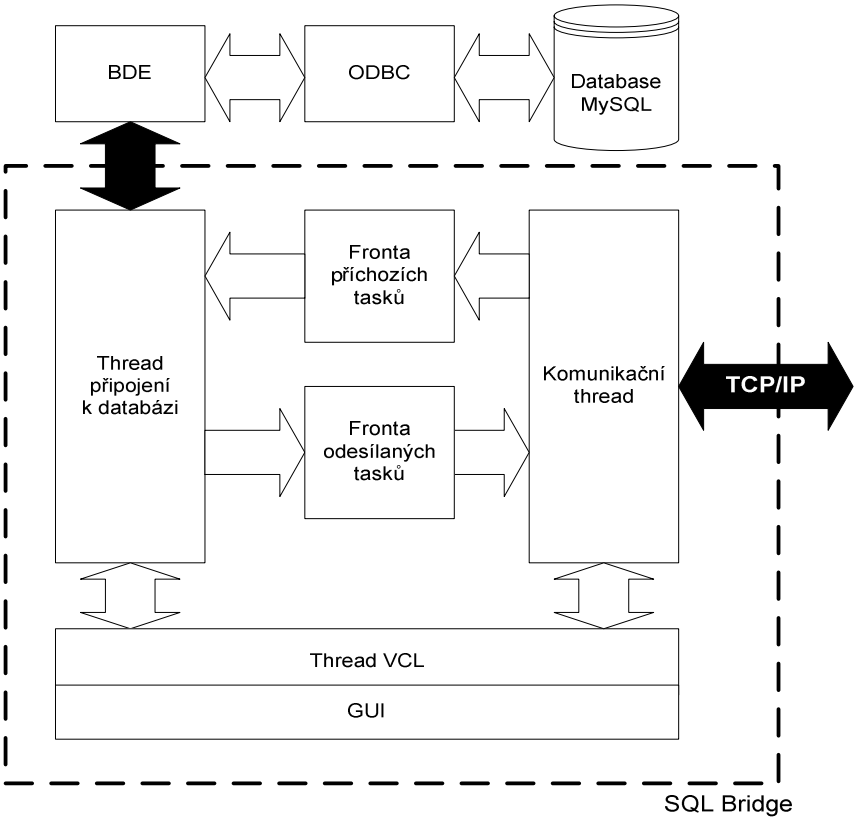
SQL Bridge zajišťuje ochranu před používáním databáze neautorizovanými osobami. U každého příkazu, který SQL Bridge dostane, jsou prověřena přístupová práva odesílajícího uživatele (viz kapitola 5.5.3.3). Pokud uživatel nemá k dané operaci s databází právo, příkaz není proveden a zpět uživateli (klientovi) je zaslána zpráva typu *ERRORMESSAGE* s chybovou hláškou.

Chybová hláška se vrací jako reakce na příkaz i v případech, že při vykonávání příkazu dojde k jakékoliv chybě. Text chybové hlášky pak chybu blíže popisuje.

SQL Bridge podporuje několik typů ukládání hodnot do databáze. Nastavení jednotlivých módů je popsáno v kap. 5.5.3.2. Jde o tři typy ukládání. Základním typem je prosté přidání naměřené hodnoty do databáze (nejčastější případ). Další je typ „DIFFERENTIONAL”, zde dojde k uložení do databáze jen v případě, že ukládaná hodnota proměnné je jiná než hodnota uložená naposledy. Posledním typem je „TIME_UPDATE“, v tomto případě můžou nastat dvě situace. Ukládaná hodnota je jiná než ta, která je poslední uložená v databázi. Poté dojde k přidání dalšího záznamu (řádku) do tabulky. Pokud je hodnota stejná, pouze se liší čas naměření proměnné, nepřidá se

řádek, ale přepíše se čas v odpovídajícím záznamu v tabulce. Tyto volby jsou výhodné pokud je dat hodně a je snaha co nejvíce komprimovat velikost databáze.

5.5.2. Architektura programu SQL Bridge



Obrázek 5.5.1: Schéma architektury SQL Bridge

Princip činnosti SQL Bridge je následující. Pokud thread připojení k databázi nalezne příkaz ve frontě příchozích úloh, provede jej a výsledná data vrátí do fronty odesílaných úloh. Ty jsou poté odeslána threadem pro komunikaci na server.

5.5.3. Konfigurace

Dále následuje popis možností nastavení v SQL Bridge.

5.5.3.1. Připojení k databázi

Pro nastavení připojení k databázi je určen panel vlevo nahoře „Connection to Database“ (Obrázek 9.3.4: Hlavní okno SQL Bridge). Zde se nastavuje uživatelské jméno a heslo pro připojení k databázi a alias databáze nakonfigurovaný v BDE, nebo ODBC manageru. Nakonfigurovaná databáze musí mít strukturu tabulek jak bylo popsáno výše v návrhu databáze. Připojit se lze na jakoukoli databázi, ke které jsou ovladače pro BDE či ODBC.

5.5.3.2. Typ ukládání

Panel vpravo dole s nadpisem „PutVar setings“ určuje, který typ ukládání proměnných do databáze (viz odstavec 5.5.1.2) se používá jako defaultní. Pokud je v úloze „PutVar“ specifikováno jinak, použije se typ požadovaný.

5.5.3.3. Nastavení práv uživatelů

Práva uživatelů pro přístup do databáze se konfigurují v externím souboru „UserAccess.txt“. Struktura souboru je následující:

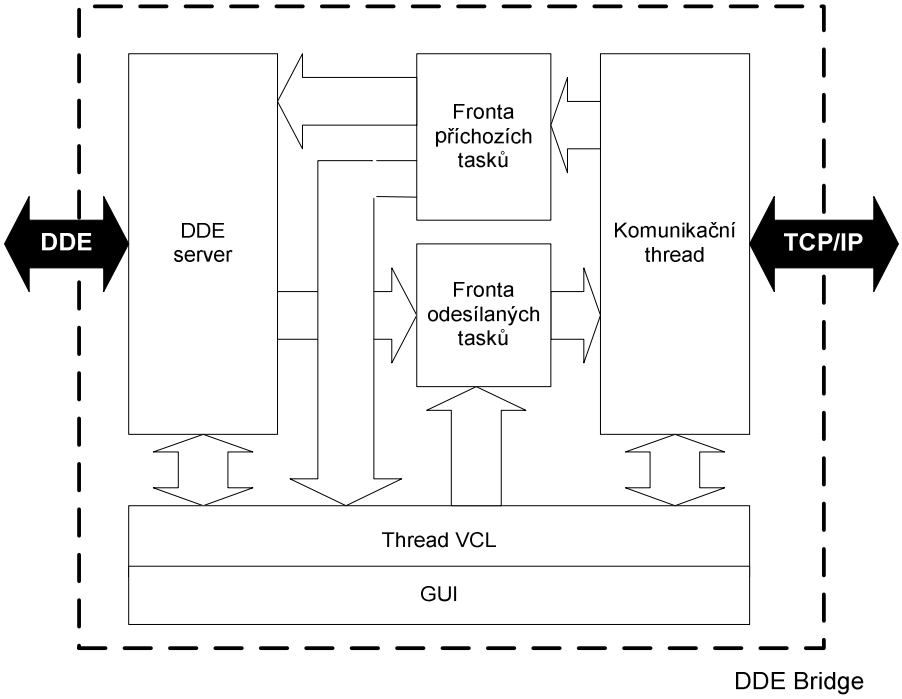
```
Username=SQLQuery, GetVar, PutVar, ClearVar, AddVar, DelVar
Karel=+,+,+,+,+,-
Pepik=-,-,-,+,+
Tonik=+,-,-,+,+
Alena=+,+,-,+,-
```

Každý řádek začíná uživatelským jménem a následuje rovnítkem, dále následuje seznam povolených „+“ a nepovolených „-“ operací s databází, pro daného uživatele, oddělený čárkami. Tato konfigurace je načtena při startu programu anebo stiskem tlačítka „Refresh access list“.

5.6. DDE Bridge

DDE Bridge je svou podstatou podobný klient jako byl SQL Bridge s rozdílem, že zde jde o napojení sítě UniServer na standardní rozhraní Windows DDE. Strana DDE rozhraní obsažená v DDE bridge je server. Nadefinované proměnné výměny dat DDE jsou při změně jejich hodnoty přes rozhraní DDE automaticky zasílány cílovému klientovi v síti UniNetwork. Naopak při zápisu dat (hodnot proměnách) ze sítě UniNetwork je jejich aktualizace oznámena všem DDE klientům napojených na DDE server v DDE Bridge. Obrázek hlavního okna a seznam odporovaných příkazů je v příloze 9.3.4.

5.6.1. Architektura programu DDE Bridge



Obrázek 5.6.1: Architektura DDE Bridge

5.6.2. Konfigurace

Veškerá konfigurace programu DDE Bridge je v hlavním okně. Do položky označené „Variable name“ se vyplní jméno proměnné v síti UniNetwork. Toto jméno je zároveň použito jako jméno aktivovaného „topic“ pro DDE server. Jsou z něho však odstraněny znaky „:“ a „\“ a nahrazeny podtržítkem. Ke jménu proměnné je ještě třeba specifikovat její typ v políčku hned vedle. Pak lze stiskem tlačítka „Add“ přidat položku do seznamu poskytovaných „DDE topic“. V případě použití tlačítka „Add to auto update“ je rovněž vytvořeno patřičné „DDE topic“, ale hodnota položky je neustále cyklicky čtena ze své adresy v UniNetwork v intervalu „Update time“. Zaškrtávací políčko „Allow poke“ rozhoduje o tom, zda v případě, že hodnota proměnné je změněna přes rozhraní DDE je tato její nová hodnota zaslána do sítě UniNetwork.

5.6.3. Připojení na data DDE Bridge

DDE klient je zapotřebí nakonfigurovat takto
DDE Service: „DDEBridge“
DDE Topic: „Data“
DDE Item: jméno proměnné

Příklad v MS Excel: =DDEBridge|Data!TestSignals_0_Triangle1

5.7. DDE Client

Jedná se o podobný program jako je DDE Bridge, avšak zde je rozhraní DDE implementováno jako DDE client. Tedy souhrnně řečeno, program DDE Client napojuje síť UniNetwork na DDE server (oproti tomu program „DDE Bridge“ napojuje UniNetwork na DDE client). Program je vhodné použít tam, kde data, která chceme zpřístupnit v UniNetwork jsou přístupná na DDE serveru. To můžou být například data ze sběrnice nebo z PLC. Seznam podporovaných příkazů je v příloze.

5.7.1. Architektura programu DDE Klient

Architektura je stejná jako u DDE Bridge ale vnitřní rozhraní DDE je DDE client.

5.7.2. Konfigurace

Nejprve je nutné připojit se na požadovaný DDE server. Toto spojení se nastavuje na panelu umístěném vlevo na hoře. Po vyplnění údajů se tlačítkem „Add“ přidá do seznamu připojených serverů. Poté lze pro jednotlivé DDE servery specifikovat DDE items, ke kterým se chceme připojit. V konfiguraci DDE item se také nastavuje pod jakým jménem bude tato proměnná zpřístupněna do UniNetwork a na jakou adresu se bude zasílat její hodnota při aktualizaci na DDE serveru. Přepínaní toho, zda bude takto její hodnota zasílána do UniNetwork se děje tlačítkem „Advice“ – „Unadvice“.

V souvislosti s tímto programem bych rád upozornil na to, že komponenty pro komunikaci po DDE dodávané jako součást programu C++ Builder jsou nefunkční při použití více jak jednoho DDE topic v aplikaci. Vytvořil jsem proto komponenty nové, funkční, které se z hlediska uživatele používají stejně jako komponenty Borlandu, ale fungují spolehlivě.

5.8. OPC Bridge

Již z názvu je patrné, že jde opět o program sloužící jako rozhraní mezi sítí UniNetwork a serverem OPC (OLE for Process Control). Na serveru OPC jsou přístupná data získaná přímo z technologií a tedy OPC bridge je velmi důležitá část celého systému. Tento klient sítě je určen pro výměnu dat mezi sítě UniNetwork s řídicími automaty a periferiemi. Je však možné jej použít i pro přístup k datům na OPC serverech určených pro simulační a testovací účely.

Na nakonfigurované položky serveru OPC je možné přistupovat standardními příkazy „GETVAR“ a „PUTVAR“. Pokud je tak nastaveno, při každé aktualizaci proměnných na serveru OPC jsou nová data automaticky zaslána na příslušnou adresu v UniNetwork.

5.8.1. Architektura programu OPC Bridge

I zde je architektura programu velmi podobná struktuře DDE Bridge (Obrázek 5.6.1), a proto zde nebudu obrázek uvádět. Jedná se opět o

propojení dvou rozhraní a uživatelského interface. Rozdíl oproti schématu DDE Bridge je v záměně části programu tvořící rozhraní DDE za rozhraní pro připojení k OPC serveru.

5.8.2. Konfigurace

Na úvodní obrazovce se po zpuštění objeví záložka „Server connection“. Dole v okně je panel „OPC Server items“, který je zatím prázdný ale po připojení k OPC serveru zobrazuje všechny dostupné proměnné serveru. Pro připojení na OPC nejprve musíme nastavit adresu uzlu sítě, kde se nachází OPC server („Node addres“). Server může být na jakémkoliv uzlu sítě, se kterým je možné a povolené propojení po standardu Microsoft DCOM. Po stisku tlačítka „Read server names“ se zobrazí seznam dostupných serverů a stiskem „Connect“ se klient připojí. Na OPC serveru je nutné nadefinovat skupiny, do kterých chceme proměnné sdružovat. Každá skupina má svoje specifické vlastnosti. Vytváření skupin a nastavování jejich parametrů se děje na kartě „Server groups“ na záložce „Group“.

Když máme vytvořenou alespoň jednu skupinu můžeme do ní začít přidávat proměnné. To uděláme na panelu dole v okně, kde najdeme ve stromové struktuře příslušnou položku, vybereme skupinu a položku přidáme. Zatrhávací políčko „Active“ určuje, zda je položka ne serveru monitorována, či je prozatím jen vytvořena.

Vytvořenou strukturu proměnných a skupin je možné procházet na kartě „Server groups“. Při kliknutí na položku struktury jsou napravo na panelu zobrazeny podrobnosti, které je možné měnit. Z těchto nastavení stojí za zvláštní zmínku zaškrtávací políčko „Sync IO“. Zde se definuje přesná interpretace příkazu „GETVAR“ serverem. Pokud je políčko zaškrtnuto, dochází při vykonávání příkazu k přímému čtení hodnoty proměnné ze zařízení. Pokud zaškrtnuto není, čte se hodnota z vnitřní cache, která je obnovována v intervalu nastaveném pro příslušnou skupinu proměnných. Příkazem „PUTVAR“ dochází vždy přímo k fyzickému zápisu do zařízení.

Funkce poskytované do sítě UniNetwork jsou, kromě čtení a zápisu proměnných, procházení strukturou dostupných proměnných serveru OPC a strukturou vytvořených skupin a položek.

5.9. OPC HDA Bridge

OPC HDA Bridge je dalším ze zástupců interfacu systému UniNetwork. Tentokrát jde o připojení na server OPC Historical Data Access. Klienti sítě UniNetwork mohou používat všechny funkce implementované OPC HDA Serverem a číst všechny jeho konfigurační a informační položky. OPC HDA server a knihovny pro jeho připojení, které byly použity, pocházejí od Tomáše Svobody z jeho diplomové práce „Průmyslová automatizace s využitím OPC“.

OPC HDA Bridge je jediný klient UniServeru určený pro výměnu dat, kde nejsou implementovány standardní funkce „GETVAR“ a „PUTVAR“. Je tomu tak, protože tyto funkce nemají pro výměnu dat po OPC HDA smysl. Zde se jedná zejména o načítání více záznamů hodnot proměnné z jejího

historického vývoje. Seznam podporovaných příkazu je opět uveden v příloze a stejně tak i pohled na hlavní okno aplikace (příloha 9.3.7).

5.9.1. Architektura programu OPC HDA Bridge

Vnitřní struktura programu je opět podobná architektuře použité pro OPC Bridge. Proto zde nebudu schéma uvádět.

5.9.2. Konfigurace

Zde, rovněž jako u OPC Bridge, je konfigurace v porovnání s ostatními klienty složitější. Můžeme říct, že část konfigurace je obdobná jako u konfigurace OPC Bridge. Je to hlavně výběr a připojení OPC HDA Serveru, pro který platí stejná pravidla a postupy jako u klienta OPC Bridge.

Hlavní rozdíl je při nastavování položek, které chceme ze serveru OPC HDA číst. Narozdíl od OPC Bridge zde se nevyžaduje definování skupin proměnných. Proměnné jsou přidávány do jedné společné skupiny. Maximální počet proměnných je deset. Každá nakonfigurovaná položka má soubor svých atributů. Ten je zobrazován vpravo od seznamu položek společně s jejich slovním popisem a hodnotami. V tomto klientu je navíc také záložka „Server info“ poskytující informace o všech agregačních funkcích serveru. Každá funkce má rovněž zobrazen svůj slovní popis.

Přehled všech těchto nastavení a konfigurací je poskytován přes rozhraní sítě UniNetwork.

5.10. Matlab Bridge

Aplikace Matlab Bridge je součástí systému, která umožňuje prostřednictvím sítě UniNetwork ovládat všechny důležité funkce aplikace Matlab. Seznam podporovaných příkazů obsahuje příloha 9.3.8.1.

5.10.1. Popis funkcí, možností a zabezpečení Matlab Bridge

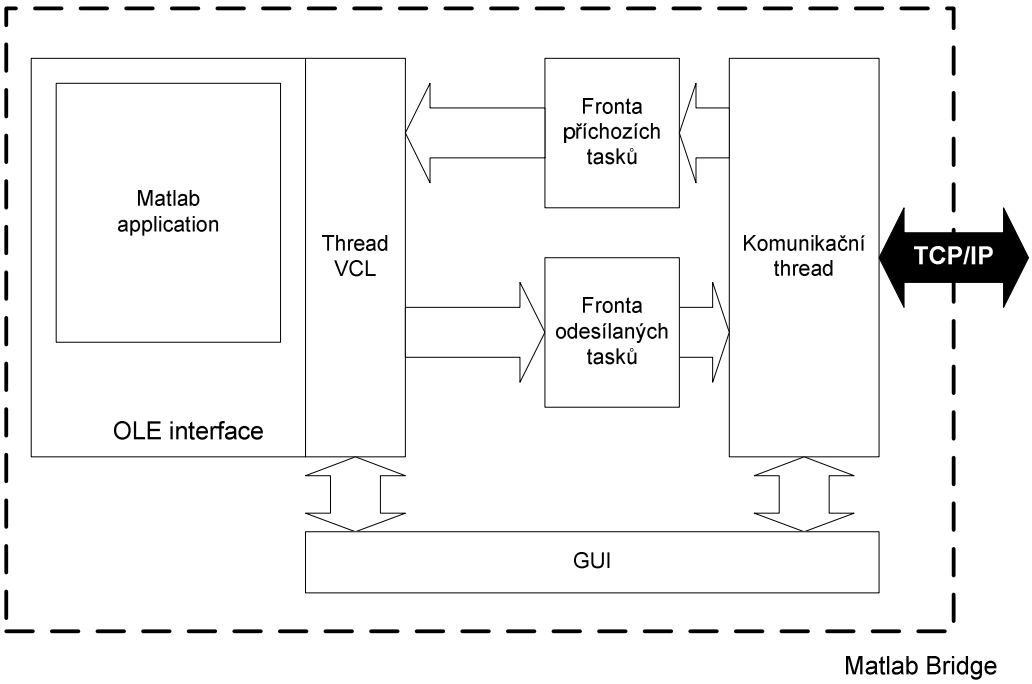
Přes síť UniNetwork lze v aplikaci Matlab spouštět soubory skriptů a funkcí, dále používat všechny interní funkce Matlabu. Komunikace probíhá tak, že klient zašle na adresu MatlabBridge příkaz „MATLABDOCOMMAND“, ve kterém je uložený příkazový string pro vykonání v Matlabu. Po provedení všech příkazů klient dostane zpět odpověď, ve které je opět jako string uložená hláška Matlabu o provedení či neprovedení příkazu.

Matlab Bridge dále podporuje příkazy pro přímé čtení a zápis matic proměnných z Matlab Environment. Jsou to příkazy „MATLABGETARRAY“ a „MATLABPUTARRAY“. I v tomto klientu jsou podporovány příkazy „GETVAR“ a „PUTVAR“ pro čtení proměnných Matlabu. Pro čtení a zápis textových řetězců, v Matlabu implementovaných jako matice prvků typu char slouží

modifikace minulých příkazů „MATLABGETARRAYCH“ a „MATLABPUTARRAYCH“.

Zajišťování bezpečnosti přístupu uživatelů je realizováno na několika úrovních. Zaprvé blokováním vykonávání příkazů, které jsou uvedeny v souboru „FCmds.txt“. V něm je umístěn seznam příkazů a jmen proměnných, které jsou pro používání v příkazech uživatelů zakázány. Seznam je uložen v textové formě je tedy jednoduché jej upravit podle požadavků. Druhý způsob zajišťování bezpečnosti je umožnění zamknout klienta, který pak může používat pouze daný uživatel (příkazy „LOCKCLIENT“ a „UNLOCKCLIENT“). A třetí způsob je vytvoření zvláštního adresáře pro každého uživatele. Který si sám může nahrávat své soubory a pracovat s nimi v Matlabu. Jiný uživatel k těmto souborům přístup nemá.

5.10.2. Architektura programu Matlab Bridge



Obrázek 5.10.1: Architektura Malab Bridge

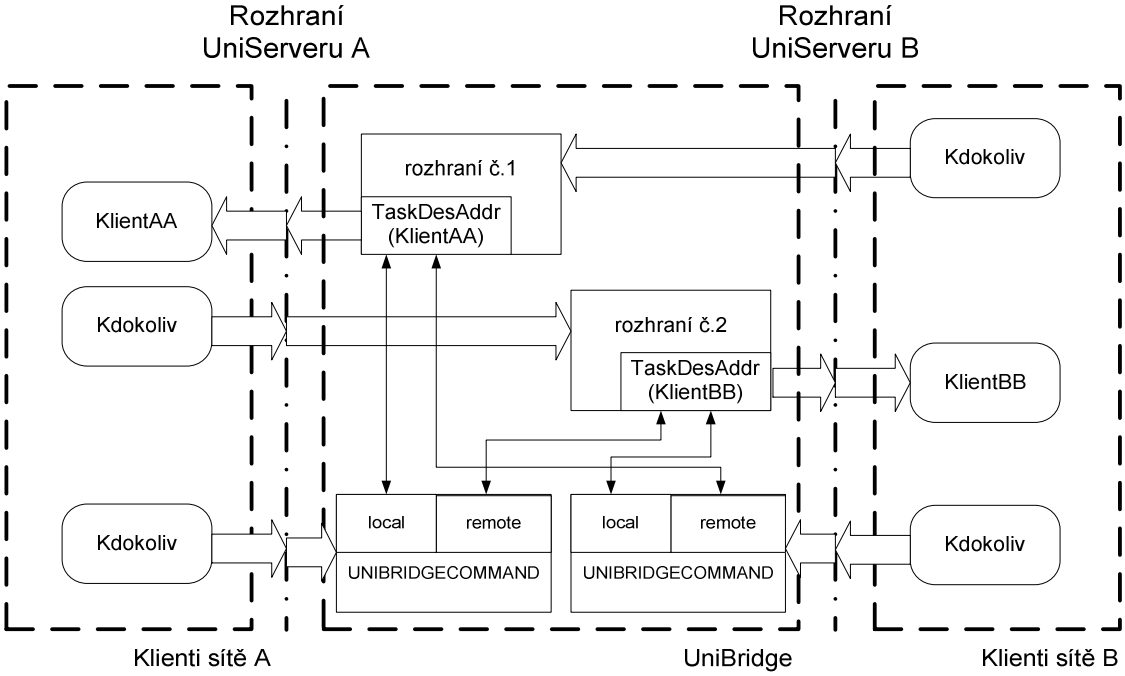
Na obrázku výše je vidět vnitřní uspořádání Matlab Bridge, kde aplikace Matlab je použita jako připojený OLE (Object Linking and Embedding) objekt (popsáno v [13]). Jeho obsluha je zajištěna threadem VCL. Po přijetí příkazu je tímto threadem spuštěno jeho vykonávání v Matlabu a výsledek je odeslán zpět na adresu zasílajícího klienta.

Ve zdrojovém kódu programu byl vytvořen modul, který po přilinkování k programu jednoduše umožňuje používání aplikace Matlab v programech v jazyce C++. Jméno modulu je „MatCon“ a nachází se na přiloženém CD s programy.

5.11. UniBridge

Propojování dvou sítí navzájem je v systému UniNetwork realizováno pomocí klienta typu UniBridge. Tento klient obsahuje dvě naprosto identická síťová rozhraní, mezi kterými přenáší pakety. Pokud na adresu jednoho síťového rozhraní UniBridge přijde nějaký paket, odešle se z druhého síťového rozhraní do vzdálené sítě. Adresu ve vzdálené síti, na kterou se budou pakety posílat lze nastavit na hlavním okně aplikace (příloha Obrázek 9.3.10). Jediný paket, který je zpracováván, je paket s příkazem „UNIBRIDGECOMMAND“. Ten v sobě nese další příkaz, určený pro UniBridge. Může jít například o příkaz ke zjištění seznamu klientů ve vzdálené síti, nebo o seznam dalších propojení vzdálené sítě s „ještě vzdálenějšími“ sítěmi. Pomocí těchto příkazů lze libovolně zjišťovat a procházet celou architekturou propojených sítí UniNetwork.

Typ propojení sítí, který je možné pomocí UniBridge, nebo vícenásobného použití UniBridge, je naprosto libovolný. Lze realizovat propojení do hvězdy, do kruhu či libovolnou kombinaci obou. Fakt, který toto umožňuje je, že mezi libovolnými dvěmi sítěmi (servery UniNetwork) lze umístit neomezený počet klientů UniBridge. A proto architektura sítě je zcela libovolná. Je ale třeba si uvědomit, že z filosofie adresování v síti UniNetwork vyplývá nutnost použití zvláštního UniBridge pro realizaci každého přenosového kanálu mezi klienty v rozdílných sítích. Jedna instance programu UniBridge, připojená mezi dvě sítě, může v dané konfiguraci tedy propojovat pouze jednoho klienta ze sítě A s jedním klientem sítě B. Pro propojení na jiného klienta se musí UniBridge překonfigurovat (buď ručně v okně aplikace nebo pomocí příkazu UNIBRIDGECOMMAND typu SETREMOTETASKDESADDR popř. typu SETTASKDESADDR)

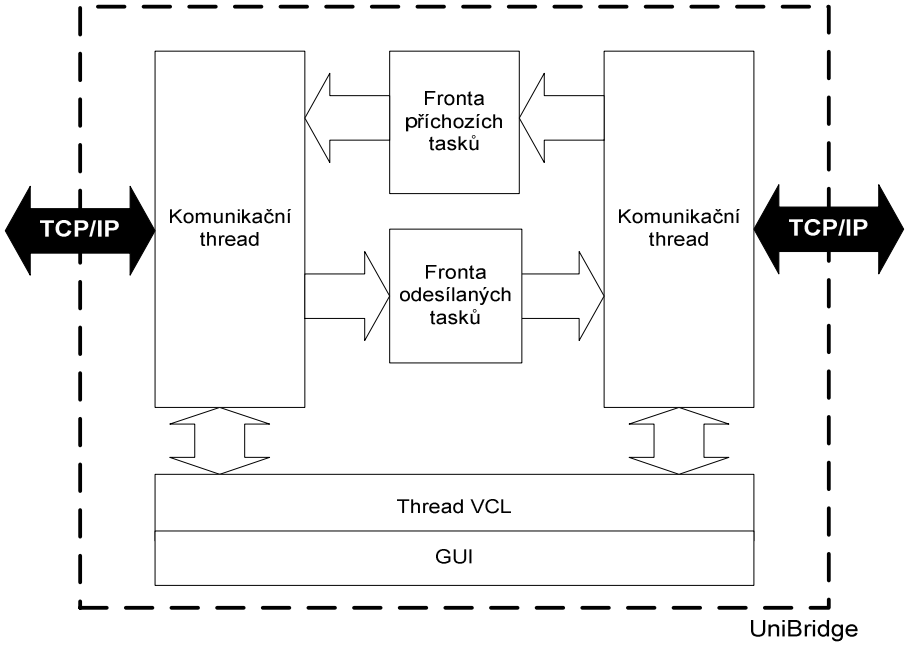


Obrázek 5.11.1: Princip UniBridge

5.11.1. Konfigurace

Konfigurace nastavení UniBridge je jednoduchá a spočívá v několika krocích. Nejprve je nutné se připojit, standardním způsobem použitým i v ostatních klientech, do obou propojovaných sítí (na oba servery). Dále je nutné nastavit string unikátně identifikující daný UniBridge. Každý připojený UniBridge kdekoliv v propojovaných sítí musí mít své unikátní jméno. Toto jméno je automaticky generováno při spuštění UniBridge, ale v případě potřeby je možné jej změnit. Posledním krokem je nastavení adres klientů, kam se budou v jednotlivých sítích posílat všechny úlohy, které na UniBridge přijdou z opačného síťového rozhraní.

5.11.2. Architektura programu UniBridge



Obrázek 5.11.2: Architektura UniBridge

5.12. File Man

File Man (zkrácenina file manager) je program umožňující správu souborů po síti UniNetwork. Uživatel tak má k dispozici všechny základní funkce pro práci se soubory jako je download, upload, mazání. Také je možno na vzdáleném počítači číst, vytvářet a mazat adresářové struktury.

Pro každého uživatele je automaticky vytvořen jeho vlastní adresář. Jednotliví uživatelé mají definovaná práva pro práci s klientem FileMan v souboru „UserAccess.txt“. Struktura souboru je obdobná jako u nastaveni SQL Bridge v kapitole 5.5.3.3. Práva které může uživatel obdržet jsou:

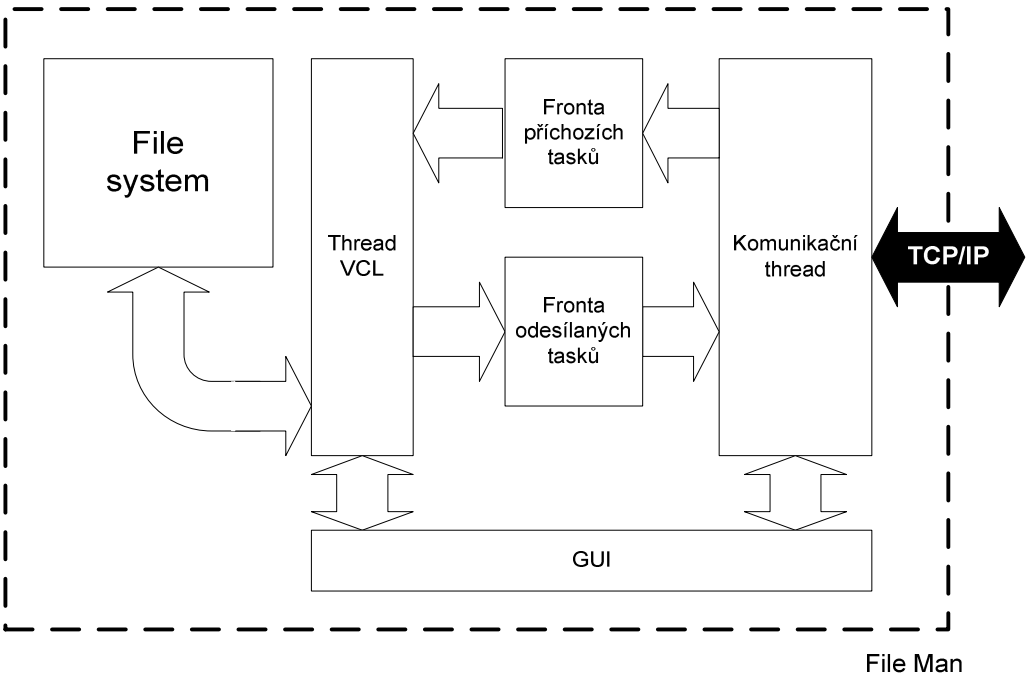
CD	povolení měnit aktivní adresář
ReadAll	povolení na download všech souborů
WriteAll	povolení na upload všech souborů
ReadHome	povolení na download souborů v domovském adresáři
WriteHome	povolení na upload souborů v domovském adresáři
MkDirAll	povolení na vytváření podadresářů všude
MkDirHome	povolení na vytváření podadresářů v domovském adresáři
DelAll	povolení na mazání všech souborů
DelHome	povolení na mazání souborů v domovském adresáři
RmDirAll	povolení na rušení všech adresářů
RmDirHome	povolení na rušení podadresářů v domovském adresáři

Tabulka 5.12.1: Tabulka práv uživatelů klienta File Man

Tohoto klienta je vhodné kombinovat s klientem Matlab Bridge. Pak uživatel může transportovat do adresáře Matlabu svoje vlastní skripty a datové soubory, se kterými potom může v Matlabu pracovat.

Seznam příkazu pro ovládání FileMan je v příloze 9.3.10.

5.12.1. Architektura programu File Man



Obrázek 5.12.1: Architektura File Man

5.13. Test Signals

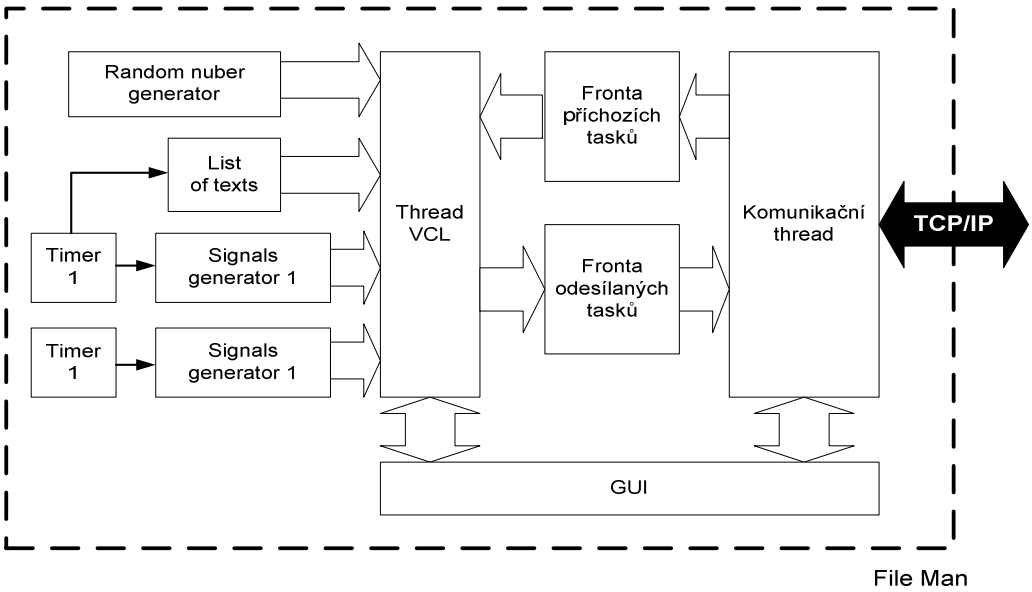
Tento klient systému je určen zejména pro testovací a simulační účely. Klient poskytuje ke čtení několik proměnných o různých typech. V programu jsou neustále generovány průběhy signálů tvaru pila, trojúhelník, obdélník. Typy proměnných těchto průběhů jsou „INT“ a u proměnných s písmenem „F“ na začátku typu „FLOAT“. Dále jsou zde dvě proměnné typu „STR“, které obsahují neustále se měnící text. Jediný příkaz, který je klientem podporován je příkaz „GETVAR“.

Všechny proměnné lze rozdělit do čtyř skupin. První dvě skupiny jsou průběhy generované časovačem 1 a časovačem 2. Frekvence, resp. periody obou časovačů se nastavují na hlavní obrazovce. Třetí skupina jsou signály jejichž hodnoty v čase se mění pouze po jejich přečtení. A čtvrtá skupina jsou proměnné náhodných hodnot.

`Triangle1, Triangle2` `FTriangle1, FTriangle2`	průběh signálu trojúhelníkového tvaru
`Saw1, Saw2` `FSaw1, FSaw2`	průběh pilovitého trojúhelníkového tvaru
`Square1, Square2`	průběh signálu obdélníkového tvaru
`Random1, Ramdom2` `FRandom1, FRamdom2`	náhodné hodnoty proměnných
`Text1, Text2`	náhodné části textu
`AutoTriangle, AutoSaw, AutoSqare` `FAutoTriangle, FAutoSaw`	průběhy signálů nejsou generovány časovačem ale posun v průběhu signálu se děje jeho přečtením

Tabulka 5.13.1: Seznam proměnných programu Test Signals

5.13.1. Architektura programu Test Signals

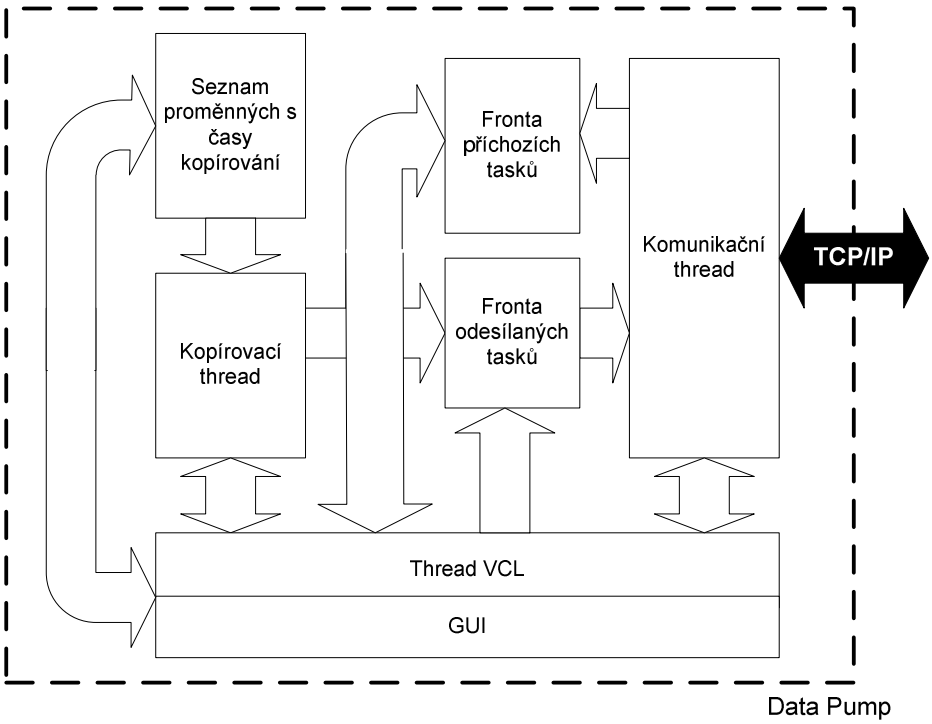


Obrázek 5.13.1: Architektura Test Signals

5.14. Data Pump

Pokud chceme zaručit nějaký stálý tok dat v systému, tedy vytvořit kanál, kterým se bude neustále posílat hodnota nějaké proměnné, máme dvě možnosti. Buď opakované posílání lze nastavit přímo na klientovi, který data obsahuje a umí je cyklicky nebo po jejich změně posílat na specifikovanou adresu v síti UniNetwork. Pokud tyto funkce uvažovaný klient neobsahuje, je jej nutno nějak stimulovat. K tomu slouží právě klient se jménem DataPump. Tento klient je schopen v nastavených časových intervalech zasílat žádosti o hodnotu proměnné („GETVAR“) na určeného klienta. Avšak jako adresa zdroje příkazu se uvede klient, na který data chceme poslat. Tímto způsobem klient, který je dotazovaný na požadovaná data ani nepozná, že o data žádá DataPump a rovnou jejich hodnotu zašle („PUTVAR“) na adresu uvedenou v žádosti jako adresu zdroje.

5.14.1. Architektura programu Data Pump



Obrázek 5.14.1: Architektura Data Pump

5.14.2. Konfigurace

Konfigurace DataPamp je velmi jednoduchá. Do políčka „Source variable address“ se jméno požadované proměnné spolu s její adresou (dle 5.2.1 Konvence pojmenovávání proměnných) např. „TestSignals:0\Triangle1“. Cílová adresa proměnné spolu s jejím jménem, pod kterým má na cílovou adresu přijít, se napíše do políčka „Destination variable address“, např.

„UniClient:0\PrBar2“. Časový interval, ve kterém se má proměnná kopírovat mezi klienty, se uvede do „Copy time distance“. Pole s informací o typu proměnné je pouze informativní pro uživatele, který DataPump konfiguruje. Poslední operací je vyplnit políčko „First copy time“ časem, kdy se má spustit kopírování proměnné. Při kliknutí do políčka je do něj načten aktuální čas. Pak již stačí stiskem tlačítka „Add“ přidat nakonfigurovanou položku do seznamu stimulovaných datových kanálů. Tlačítko „Copy now“ provede stimulaci kopírovaní vybrané položky (nahoře v okně v seznamu) okamžitě, bez ohledu na nastavenou periodu kopírování.

5.14.3. Popis činnosti programu

Rozesílaní paketů pro stimulaci přenosu pro všechny nadefinované kanály obstarává zvláštní thread. Ten cyklicky prochází seznamem nastavených stimulací a v případě, že čas, ve kterém měla být stimulace provedena, již nastal, zašle patřičný paket do sítě a nastaví čas další stimulace na hodnotu aktuálního času plus hodnotu nastavenou jako periodu stimulací. Tuto svoji činnost neustále opakuje.

5.15. Klient InOut

Klient InOut je určen především pro testovací a simulační účely. Je určen pro interakci proměnných systému s uživatelem. Jde o jednoduchou vizualizaci, která umožňuje snadno zobrazovat hodnoty proměnných v systému pomocí několika typů příkazů (seznam příkazů je v příloze 9.3.13.1). Dále je možné tímto klientem vyvolávat události v systému. V okně aplikace (Obrázek 9.3.14) je umístěna sada tlačítek označených „Button 1“ až „Button 10“. Při stisku některého z tlačítek je klientovi, který je vybraný nahoře v seznamu, zaslán příkaz „BUTTONPRESSED“, který mu oznamuje, že bylo stisknuto dané tlačítko. Jinou událostí, která je publikována do sítě klientů je zaškrtnutí některého z 10 nakonfigurovaných zaškrtávacích políček.

Ostatní políčka v okně aplikace slouží pro zápis (zápis proměnné po sítí, čtení uživatelem) a čtení jejich hodnoty. To se děle zasláním příkazu „PUTVAR“, nebo „GETVAR“ se jménem příslušné proměnné. Jména proměnných odpovídají jménům objektů v na formuláři. Jmenovitě to jsou: „Edit1“ až „Edit10“, „CheckBox1“ až „CheckBox10“ a „Memo1“ až „Memo10“.

Architektura a funkce klienta je natolik podobná předchozím architekturám a popis práce programu je natolik jemnoduchý a jasně čitelný ze zdrojového kódu, že přehledové schéma struktury aplikace nebudu uvádět.

5.16. UniClient

UniClient je vizuálním rozhraním ke všem doposud naprogramovaným klientům sítě UniNetwork. Ve většině případů jde o vizualizaci rozhraní, která

poskytují klienti po síti. Z programu UniClient lze všechny klienty ovládat v míře, kterou dovolují. Číst a zapisovat jejich proměnné a používat všechny příkazy klienty podporované. Pro většinu typů klientů je v programu umístěn zvláštní panel. Některé panely jsou však univerzálnějšího charakteru a jejich funkce není přesně vyhraněna jen na jeden typ klientů. Obrázky jednotlivých oken (panelů) aplikace naleznete v příloze Obrázek 9.3.15. Dále budu popisovat jednotlivé panely a jejich konfigurace.

5.16.1. Panel „Connection“

Na tomto panelu se nachází několik menších rámů. První rám obsahuje informace týkající se připojení k UniServeru. Nastavení připojení je stejné jako v ostatních klientech a nebudu se jím dále zaobírat. Rám „Server select“ obsahuje jména všech klientů v systému typu „Matlab:x“. Vybírá se zde který MatlabBridge bude použit pro zasílaní příkazů z panelů Matlab. Dalším rámem je „Common info“, zde se zobrazují některé obecné textové informace přicházející na UniClienta. Jde například o úlohu typu „VARSAVED“. Posledním panelem je „Server connection komponent state“. Zde je umístěn vizuální objekt komponenty realizující připojení k UniServeru. Kliknutím pravým tlačítkem se rozbalí menu s informacemi o připojení.

5.16.2. Panel „Send task“

I zde je několik oddělených částí panelu. První „Clients list“ ukazuje seznam všech klientů připojených k UniServeru. Kliknutím se vybere klient, kterému budou posílány příkazy z tohoto panelu. Rám „Send task (type 1)“ slouží k zasílání obecného příkazu jakémukoliv klientovi. Jde však pouze o příkazy, jejichž datová část se skládá z jednoho textového stringu (může se jednat i o víceřádkový). Pod položkou „Templates“ si můžeme vybrat některý ze složitějších příkazů, které při opakovaném používání nemusíme vždy znovu psát. Seznam těchto předdefinovaných textů je načítán ze souboru „UniClient.ini“. Struktura souboru je zřejmá z jeho obsahu. Po odeslání příkazu z tohoto rámu se v položce „ID of last sent“ objeví identifikační číslo rámce s odeslaným příkazem. Rám „Send task (type 2)“ se od předchozího rámu liší pouze tím, že umožňuje do odesílané úlohy vyplnit dvě datové položky typu string.

5.16.3. Panel „Send task B“

Zde se kromě seznamu klientů nachází další dva rámy. Rám nahoře „Send SQL command“ je určen pro zasílání příkazů SQL pro klienty typu „SQLBridge“. Adresa, na kterou je příkaz posílán, se vybere v políčku „Address“. Políčko „Source adrdess“ obsahuje adresu, která je vyplněna do úlohy jako adresa zdroje příkazu a přijde na ni odpověď s výsledkem SQL příkazu. V rámu vpravo nahoře si je možné vybrat, zda požadujeme odpověď v textovém či datovém tvaru (viz seznam příkazů SQL Bridge, kapitola 5.5).

Dolní rám slouží ke komunikaci s klienty „User Access Server“. Z nich je možné číst a při patřičném oprávnění i měnit práva uživatelů.

5.16.4. Panel „SQL Result“

Na tomto panelu jsou zobrazovány odpovědi na příkazy zasílané klientům SQL Bridge. Zobrazena zde může být jak odpověď datového typu (objeví se okno s tabulkou), tak i odpověď textového typu (zobrazí se jako text v okně).

5.16.5. Panel „Variables“

Na tomto panelu se nacházejí rámy pro čtení dat z databáze a z ostatních klientů. V rámu „Common settings“ se nastavuje cílová adresa příkazu a adresa kam je zaslána odpověď. Pod tímto rámem jsou umístěny čtyři tlačítka. Jde o tlačítka, které odesílají předdefinované příkazy pro SQL Bridge, které se dotazují na strukturu a seznam proměnných v databázi. Speciální význam má pak tlačítko (příkaz) „GETVARSTRUCDAT“. Po obdržení odpovědi na tento příkaz je vytvořen strom reprezentující strukturu proměnných, který je na panelu „Variables B“.

Panel „Save Data“ je formulář pro vyplnění příkazu „PUTVAR“ a jeho modifikací „DIFFERENCIAL“ a „TIMEUPDATE“. Políčka pro vyplnění se shodují co do názvu i významu s položkami příkazu „PUTVAR“.

Rám „Universal command“ je opět formulář pro zasání libovolného příkazu, tentokrát s třemi datovými položkami typu text string.

Rám „GETTIMEVAR“ je určen k odeslání stejnojmenného příkazu pro databázi. To samé platí i pro rám „CLEARVAR“, určen pro smazání záznamů proměnných v databázi.

5.16.6. Panel „Variables B“

Jde o panel, kde jsou umístěny rámy, které logicky souvisí s rámy na předchozím panelu, ale už se na něj nevešly. Jde o strom hierarchické struktury proměnných (vyplněný po předchozím obdržení odpovědi na příkaz „GETVARSTRUCDAT“). K němu patří i rám „Variable info“. Zde se při kliknutí na nějakou proměnnou ve stromu struktury zobrazují informace o proměnné (typ, popis, …). Při zaškrtnutí políčka „Auto get value“ se zde zobrazuje i hodnota proměnné.

Rám „GETVAR“ je určen pro příkaz k přečtení hodnoty proměnné. Jméno proměnné je specifikováno v „Name“. Příkaz je zaslán na adresu vyplněnou v rámu „Commnon setting“. Rám „PUTVAR“ zobrazuje data z příchozího příkazu „PUTVAR“. Rámy „Add Var“ a „Delete Var“ jsou určeny pro přidávání a rušení proměnných v databázi SQL Bridge.

5.16.7. Panel „FileMan“

Jde o vizualizaci rozhraní poskytovaného programem FileMan. Vlevo se nachází seznam klientů pojmenovaných „FileMan:“ a jeho číslo. Při kliknutí na nějakého klienta v okně je z něj načten seznam souborů v aktivním

adresáři. Ten je zobrazen v okně vpravo. Při kliknutí na soubor jsou v rámu „File info“ zobrazeny detailní informace o souboru. Na panelu jsou tlačítka umožňující všechny standardní manipulace se soubory, zahrnující např. upload a download souborů. Dále je zde umístěn seznam disků na vzdáleném systému, ze kterých je možno si vybrat. Panel se zobrazením souborů umožňuje standardním dvojklikem na adresáře procházení adresářové struktury.

5.16.8. Panel „Matlab cmd“

Zde je možné zadávat příkazy pro klienty Matlab Bridge. Příkazy jsou posílány klientovi Matlab vybranému na panelu „Connection“. Zadávat lze jak jednořádkové příkazy, tak i celé skripty (v okně v dolní části panelu). Po odeslání příkazu a obdržení odpovědi je text odpovědi zobrazen v okně na místě kde bylo okno s Matlab skriptem. Zaškrtávací políčko „Wrap“ určuje, zda budou v těchto oknech zalamovány řádky.

5.16.9. Panel „Matlab matrix“

Zde může uživatel načítat a ukládat matice z programů Matlab připojených přes Matlab Brigde. Jméno matice se vyplňuje do políčka „Name“. Vpravo nahoře se vybírá mezi zobrazením reálné a imaginární části matice. Hned vedle je zobrazena velikost matice.

5.16.10. Panel „Matlab picture“

Na tomto panelu je možné zobrazovat obrázky uložené v maticích Matlabu. Obrázky mohou být dvou typů. Černobílé, reprezentované maticí MxNx1, s prvky typu double nebo uint8 o hodnotách pixelů 0 – 255. Nebo barevné, reprezentované maticí MxNx3, s prvky typu double nebo uint8 o hodnotách pixelů 0 – 255. Jméno matice s obrázkem se nastavuje do kolonky vpravo nahoře.

5.16.11. Panel „OPC Bridge“

Zde se jedná o vizualizaci klientů typu OPC Bridge. Nejprve je nutné vybrat požadovaný klient v seznamu, pak je načtena jejich konfigurace. Lze procházet celou strukturou proměnných poskytovaných OPC serverem a strukturu definovaných skupin proměnných. Při kliknutí na proměnnou nebo na skupinu jsou informace o ní zobrazeny v rámu „Item info“ umístěném vpravo dole.

5.16.12. Panel „OPC HDA Bridge“

Tento panel je podobný předchozímu panelu OPC Bridge. Zde jde o vizualizaci rozhraní OPC HDA Bridge. Je možné získávat informace o definovaných proměnných, o jejich atributech a jejich popisu. Dále je možné číst informace o agregačních funkcích poskytovaných serverem. Na záložce „OPC HDA Server data“ lze zadávat příkazy pro čtení dat z OPC HDA serveru. Jde o tři základní příkazy „READRAWDATA“, „READPROCESSEDDATA“ a „READTIMEATTIME“. Data odpovědi jsou zobrazována na tomto panelu vpravo.

5.16.13. Panel „Demo“

Na panelu „Demo“ jsou umístěny základní vizualizační prvky typu progress bar a slider. Jejich hodnoty jsou nastavovány při přijmutí příkazů „PUTVAR“ se jménem proměnné totožným se jménem rámu s vizualizačním prvkem. Do kolonky „Net address“ je možné nastavit adresu, na kterou je zasílána hodnota nastavená na vizualizačním prvku, při její změně. Nejjednodušší situace je například pokud jsou do systému připojeni klienti „UniClient:0“ a „UniClient:1“ a na kartě klienta č.0 je v rámu „PrBar2“ nastavena síťová adresa proměnné na „UniClient:1\PrBar2“. Pak při posunu slide baru v tomto klientovi je nastavená hodnota vizualizována i v okně klienta č.1.

5.16.14. Panel „UniNetwork structure“

V okně tohoto panelu je v případě, že k serveru jsou připojeni další servery pomocí klientů „UniBridge“, zobrazena struktura celé sítě propojených serverů. Načítání a procházení strukturou má však za následek překonfigurování použitých klientů „UniBridge“. Je třeba otestovat všechny klienty napojené do sítí, to však lze provést jedině se změnou nastavení propojovacích článků sítí. Při kliknutí na jednotlivé propojovací klienty „UniBridge“ ve stromu struktury, jsou informace o jejich nastavení zobrazeny v rámu „UniBridge info“.

5.17. UniMessenger

UniMessenger je klient systému, který zobrazuje chybová a informační hlášení uživateli. Možné použití tohoto klienta je také pro protokolování událostí v systému. Všechny došlé zprávy se vkládají do okna nad sebe a pomocí rolovací lišty je možné je prohlížet. Počet řádků v okně zpráv je omezen na 2000. Po překročení tohoto limitu je část zpráv smazána.

Program podporuje i jeden speciální typ zprávy. Je to zpráva, která nese otázku pro uživatele a požaduje od ní odpověď. Tato odpověď je zaslána zpět klientovi, který otázku zaslal.

Okno klienta může být schované buď na liště aplikací, nebo v „icon tray“. Po přijmutí jakékoliv zprávy je okno aplikace zobrazeno a přesunuto do popředí. Zároveň je příchod zprávy doprovázen zvukovým signálem, aby uživatel byl na zprávu důrazně upozorněn.

Pohled na okno UniMessengeru a seznam příkazu je v příloze Obrázek 9.3.16.

5.18. UserAccesServer

Tento klient systému UniNetwork je určen pro poskytování autorizačních nastavení jednotlivých uživatelů sítě této. Jakýkoliv klient si může vyžádat záznam o přístupových právech registrovaného uživatele. Jde o obdobu uživatelských práv používaných v klientech SQL Bridge, File Man a dalších. Tentokrát jsou však tato nastavení přístupná ke čtení a nastavování po síti. Soubor, ve kterém jsou nastavení uložena se jmenuje „UserAccess.txt“ a má strukturu stejnou jako konfigurační soubor programu SQL Bridge (kapitola 5.5.3.3). Při vyžádání přístupových nastavení uživatele („GETUSERRIGHTS“) je vrácen text, který se nachází v tomto konfiguračním souboru za rovnítkem v řádku začínajícím jménem uživatele.

Text s právy uživatele lze nastavovat i po síti. Tuto možnost má jen uživatel „Administrator“, který muže pomocí příkazu „SETUSERRIGHTS“ zapíše do tabulky s uživateli.

5.19. Klient UniChat

Tento klient slouží pro konverzaci mezi uživateli (operátory) pracujícími se sítí UniNetwork. Po připojení klienta UniChat do sítě jsou načteny adresy všech ostatních klientů typu chat v síti. Každému takovému klientovi je zaslán požadavek na sdělení nick name, který si uživatel zvolil. Tyto seznamy jsou zobrazeny v okně nahoře vpravo. Nyní již může začít konverzace, text se píše do políčka dole a odesílá tlačítkem vpravo od něj. Podle stavu zaškrtávacího políčka „Send to all“ je odeslaný text doručen všem klientům UniChat, nebo pouze těm, kteří jsou vybrání v seznamu. Při uzavření klienta, nebo při odhlašování ze sítě je automaticky odeslána zpráva všem ostatním klientům, aby si obnovili seznamy uživatelů.

5.20. Klient Signal

Tento program byl vytvořen hlavně pro jednoduché vizualizace alarmů nebo událostí v systému. Program umožňuje spuštění akustického (vydávání

trvalého zvuku) nebo vizuálního (velký blikající objekt na obrazovce) alarmu a zobrazení hlášení o důvodech alarmu. Dalšími příkazy lze alarm, nebo jen některou jeho část, opět deaktivovat. Ukázka vizuálního alarmu a seznam příkazu je uveden v příloze 9.3.18.

5.21. Klient Runner

Klient s názvem Runner byl vytvořen jako ukázka kódu - skriptu v jazyce C++, který řídí toky a zpracování dat v systému. Takto jednoduše se mohou řídit nejen toky dat ale hlavně manipulace s daty a provádění výpočtů s nimi. Jako příklad bych uvedl následující problém.

Do systému je zapojen klient MatlabBridge připojený na Matlab, ke kterému je připojena video kamera snímající nějaký technologický proces. Dále je připojen ještě jeden klient MatlabBridge na jiném počítači sloužící pro provádění výpočtů. Dále je připojen klient OPCBridge, realizující čtení a zápis proměnných do technologie. Dalším klientem sítě je nějaký nástroj umožňující vizualizaci dat. Řekněme, že použijeme dva klienty, UniClient pro zobrazení obrázků a Signal pro zobrazení alarmů. Pak funkce řídicího skriptu může být například následující. Nejprve jsou zaslány příkazy Matlabu pro sejmutí obrázku z kamery a provedení jeho základního zpracování. Následně je obrázek přenesen do druhého Matlabu, kde je spuštěn algoritmus pro rozpoznávání objektů na obrázku, který je velmi časově náročný a nebude zabírat strojový čas prvního Matlabu. Ten už může mezitím zpracovávat druhý obrázek. Výsledek rozpoznávání je načten do řídicího skriptu a podle výsledných hodnot je rozhodnut akční zásah do technologie, případné spuštění alarmů pro uživatele. Podle těchto rozhodnutí jsou odeslány příslušné příkazy pro klienty OPC Bridge a Signal. Pokud je spuštěn alarm, může ještě například být zobrazen obrázek z kamey Matlabu, kde je vidět důvod k alarmu, v okně programu UniClient.

Výpis příkladu řídicího kódu je v příloze 9.3.19.1.

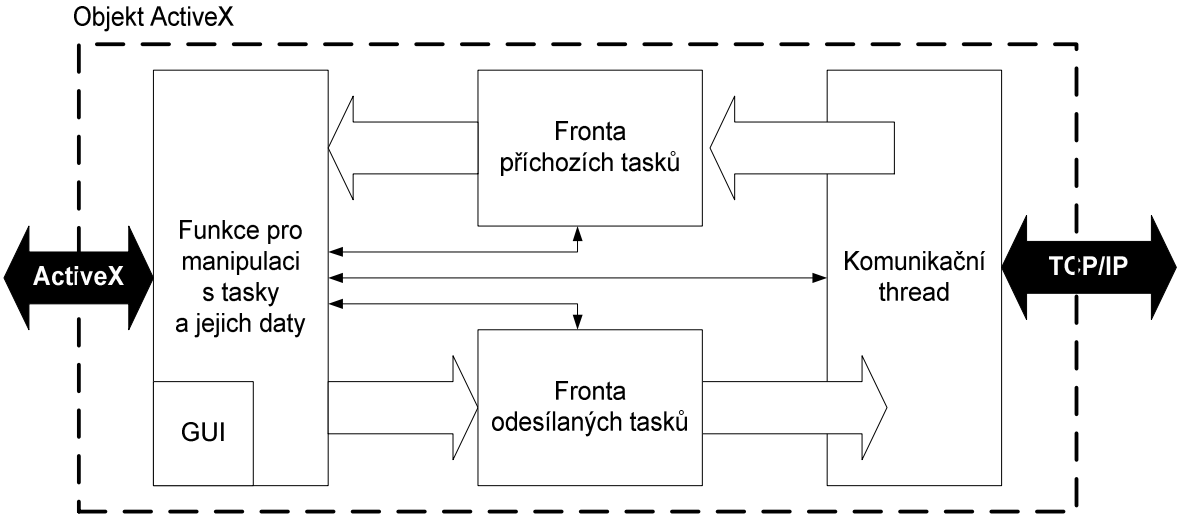
5.22. Klienti typu ActiveX

5.22.1. Objekt ActiveX poskytující připojení do sítě UniNetwork

Velmi důležitou součástí systému je objekt, který je typu ActiveX a umožňuje realizovat propojení se sítí UniNetwork. Tento prvek dovoluje všem aplikacím, které podporují technologii ActiveX přijímání a odesílání paketů do sítě UniNetwork. Aplikace pak ovládá pouze tento objekt a v síti UniNetwork se vyskytuje v postavení plnohodnotného klienta schopného veškeré komunikace. Fyzicky jde o komponentu používanou ve všech vytvořených klientech, ale nyní přeloženou s rozhraním ActiveX. Popis této komponenty byl již uveden v kapitole 5.4 Klient Builder komponenta, ActiveX klient. Soubor, který reprezentuje univerzálního klienta ActiveX se jmenuje

„UniNetworkActiveXConnectionXControl1.ocx“. Registruje se pomocí souboru „reg.bat“. Jméno zaregistrovaného ActiveX objektu je „UniNetwork.ActiveX.Connection“. V adresáři jsou přiložené i typové knihovny pro použití v programovacích jazycích. Takový objekt lze jednoduše používat z aplikací jako je MS Excel, MS Word, Matlab (nyní nikoli jako objekt OLE, ale jako samostatný klient), Internet Explorer, jazyce Visual Basic a další. Na přiloženém CD jsou ukázky jeho použití. Jsou to soubory „connect.doc“ a „connect.xls“.

5.22.1.1. Architektura univerzálního klienta ActiveX



Obrázek 5.22.1: Architektura klienta ActiveX

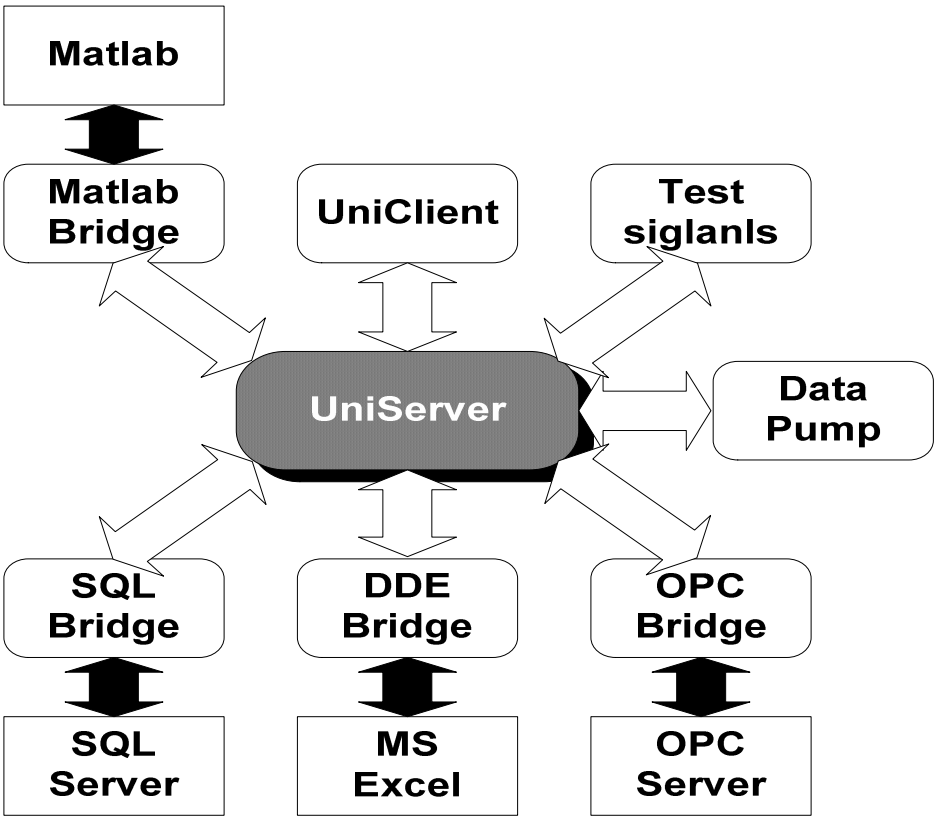
5.22.2. Další ActiveX klienti

Některé klientské aplikace sloužící pro interakci systému s uživateli byli modifikovány a přeloženy jako objekty ActiveX. Jde o program „UniClient“ a program „UniChat“. Jejich nová jména jsou „ChatActive“ a „UniClientActiv“. Výhodou těchto dvou klientů je jejich snadné umístění do webové stránky. Uživatel pak může v Internet Exploreru zadat jen adresu s umístěním takové stránky a klient je z Internetu automaticky nahrán a nainstalován. Poté se již může jen za pomoci svého uživatelského jména a hesla přihlásit do sítě UniNetwork a tak je mu umožněn plnohodnotný přístup ke všem klientům systému. Tímto způsobem je zajištěno snadné používání, správa a dohled nad systémem odkudkoliv po síti Internet. Ukázky takových jednoduchých web stránek jsou přiloženy u těchto klientů.

6. Test systému UniNetwork

Po dokončení systému bylo třeba ověřit jeho funkčnost, a proto jsem provedl následující test. Účelem testu bylo ověřit systém jednak co do stability při delším času jeho chodu a dále také při vyšším vytížení systému přenášenými daty. Realizoval jsem tedy experiment, ve kterém jsem spojil oba tyto testy dohromady.

Do testu jsem se snažil zapojit co možná nejvíce hlavních částí systému UniNetwork. Vybral jsem SQL Bridge, Matlab Bridge, OPC Bridge, DDE Bridge, UniClient, TestSignals a pochopitelně UniServer. Konfigurace systému, použitá při testu, je na následujícím obrázku.



Obrázek 5.22.1: Schéma konfigurace systému při testu

Datové přenosy (kanály) v systému byly realizovány pomocí klienta DataPump, který pravidelně vysílá řídicí signály, tedy pakety, do systému. Tyto požadavky stimulují dané klienty, aby zaslali hodnotu požadované proměnné na adresu jiného specifikovaného klienta. Šlo tedy o stimulovanou, neboli řízenou, výměnu dat (oproti změnové, nastávající při změně hodnoty proměnné, což může být v čase nepravidelně).

Datové toky v systému byly nakonfigurovány takto:

Pořadavek → Zdroj (jméno prom.) → Cíl (jméno prom)
DataPump → OPC (Triangle Waves.Real4) → SQL Bridge (Teplota T1)
DataPump → TestSignals (Triangle1) → DDE Bridge (Triangle)
DataPump → Matlab Bridge (mat_a) → SQL Bridge (U1)
DataPump → SQL Bridge (Teplota T1) → UniClient (TrBar3)
DataPump → TestSignals (Triangle1) → OPC (Bucket Brigade.Int2)

Tabulka 5.22.1: Datové toky při testu

K realizaci přenosu hodnoty jedné proměnné (jeden datový kanál) je potřeba přenesení třech paketů přes server. První je požadavek, který zašle klient DataPump pro klienta, ze kterého se data požadují. Dále se hodnota požadované proměnné zašle na cílového klienta. A třetí paket odesílá cílový klient, jako potvrzení, že hodnotu přijal. Žádosti o proměnné byly vysílány v intervalu 200 ms.

Celkem tedy je pět proměnných, každá je žádána 5krát za sekundu a na každou výměnu proměnné je potřeba přenesení třech paketů. Celkem je to 75 paketů za sekundu. Pokud vyjádříme průměrný čas mezi přenosem jednotlivých paketů je to 13,3 ms.

Systém v této konfiguraci jsem nechal pracovat tři dny bez jakéhokoliv vnějšího zásahu a během této doby nedošlo k žádné chybě, k žádnému výpadku dat ani k žádnému přerušení spojení s žádným klientem. Během této doby bylo v systému přeneseno celkem cca 19 milionů paketů. Při pokusu jsem sledoval využití operační paměti jednotlivými klienty a serverm. Hodnoty na začátku i na konci pokusu se shodovaly. V systému tedy nedochází k „protékání“ paměti. Tento fakt je nutným předpokladem pro stabilitu jakéhokoliv systému.

Na základě souhrnu těchto faktů lze předpokládat, že ani při delším provozu systému by nemělo docházet k chybám a lze tedy prohlásit, že systém neobsahuje žádné zjevné chyby, bránící funkci či provozu, a co do vlastního chodu je spolehlivý a stabilní.

7. Shrnutí, závěr a možnosti dalšího vývoje systému

Ve své diplomové práci jsem navrhl a zrealizoval systém programů společně tvořící distribuovaný systém umožňující sběr a výměnu dat mezi automatizačními prvky technologických procesů (sběrnice, řídicí automaty PLC), ukládání těchto dat do databázových struktur a umožňující využití těchto dat pro řízení na vysoké úrovni řídicí hierarchie. Systém se skládá z mnoha částí, které jsou navzájem propojeny komunikačním protokolem TCP/IP po síti Internet. Příklady těchto částí systému, které se mohou aktivně podílet na procesu řízení jsou i programy MathWorks Matlab, MS Excel, Factory Suite 2000 InTouch, a další. Díky možnosti ukládání procesních dat do databáze je možné v řídicích algoritmech používat i data z delšího časového období a dosahovat tak ještě vyšší efektivitu řízení. Současně zakomponování kvalitních vizualizačních nástrojů do systému, jako je např. program InTouch, umožňuje realizovat kvalitní a přehledné vizualizace dat v systému.

Byl vytvořen server a k němu 20 typů klientů připojitelných do systému. Tento způsob realizace zaručuje značnou modulárnost a modifikovatelnost systému pro různé účely a různé typy řešených úkolů. Jedním z vytvořených klientů je klient, který je standardu ActiveX a ještě dále tak rozšiřuje možnosti přístupu do systému ze všech programů podporujících tento standard.

Systém byl testován na stabilitu v nepřetržitém provozu po dobu tří dní. V testovací konfiguraci byly systémem přenášeny informace v intervalech v průměru 13 ms mezi 7 různými klienty. Za celou dobu v systému nedošlo k jedinému selhání a je tedy možné systém prohlásit za stabilní co se týče výpadků přenosu či selhání celého systému.

Podařilo se mi splnit všechny požadavky, které byly v úvodu na systém kladeny a všechny cíle které jsem se rozhodl realizovat.

7.1. Možnosti dalšího vývoje systému

Vzhledem k faktu, že struktura celého systému včetně struktury jeho jednotlivých programových částí byla v rámci tohoto textu podrobně popsána a vzhledem k tomu, že zdrojové kódy ke všem programům a jejich částem, které byly vytvořeny, jsou k této diplomové práci přiloženy a ponechány k volnému použití, je umožněn další rozvoj systému podle budoucích požadavků.

V současném stavu je systém UniNetwork jednoduše a výhodně použitelným pro realizaci výměny dat mezi různými systémy, pro které byly realizovány klientské aplikace (rozhraní). Data přenášená systémem UniNetwork lze rovněž snadno archivovat na databázovém serveru MySQL. Použitelnost systému v praxi se ještě zvýší pokud budou naprogramováni další klienti UniServeru, kteří propojí UniNetwork s dalšími fyzickými zařízeními. Jako jednoduchý příklad by se mohlo jednat o podporu sériového portu a následně propojení se zařízením připojeným na tento port. Tím by

mohla být například myš, kterou by tak bylo možno ovládat rameno nějakého robota. Nebo modely systémů, připojené na sériový port, které by bylo možno ovládat daty z UniNetwork, potažmo např. z Matlabu. Další užitečné propojení by byla realizace klientů, kteří by mohli číst přímo data z dalších druhů sběrnic (ModBus, ProfiBus, …), nebo přímo z PLC bez použití dalších rozhraní jako je OPC server.

Zajímavým rozšířením systému UniNetwork by byla realizace zařízení založeného na jednočipu, který má implementované rozhraní TCP/IP. Pak by napojení tohoto obvodu k síti UniNetwork bylo velmi jednoduché a umožnilo by velmi snadno realizovat ovládání obvodu přes Internet.

V praxi použitelné by mohlo také být vytvoření klientů systému UniNetwork pro jiné platformy. Jako příklad uvádím systémy Unix a hlavně RT OS WxWorks. Takto by bylo snadné získávání dat pro tyto systémy například z MS Excelu, Matlabu, digitální kamery, databáze, atd.

Asi nejzajímavějším rozšířením systému UniNetwork by bylo vytvoření aplikace schopné zpracovávat řídicí struktury ve tvaru Petriho sítí, nebo žebříčkové reléové struktury, kde vstupní a výstupní proměnné v těchto diagramech jsou proměnné sítě UniNetwork fyzicky umístěné na dalších klientech v síti. Tyto aplikace by byly schopné řídit předávání dat mezi klienty systému, porovnávat a zpracovávat jejich hodnoty a rozhodovat o dalším vývoji řízení. Jinou modifikací této myšlenky je vytvoření jakéhosi vlastního řídicího jazyka, který by řídil výměnu dat mezi klienty. Skript vytvořený v tomto jazyce by bylo možno jednoduše přeložit do jazyka C++, nebo pomocí něj interpretovat. Struktura programu v C++ by se pak velmi podobala kódu programu „Runner“, který byl popsán výše v textu a je uveden v příloze. Překlad na obdobnou strukturu programu by bylo nutné provádět i v případě realizace řízení zmiňovanými Petriho sítěmi nebo žebříčkovou strukturou.

Výše popsané aplikace systému UniNetwork by byly realizací řídicího systému na velmi vysoké úrovni hierarchie řízení, která je jednoduchá na programování a vytváření řídicích algoritmů. Celý systém by byl pak schopný koordinovat mnoho procesů navzájem mezi sebou pomocí zápisu takto složitého řízení v jednoduchých strukturách, například Petriho sítích.

Vhodným doplněním současného balíku programu UniNetwork je zdokonalení webového rozhraní systému. Spočívalo by v napojení UniNetwork na nějaký webový server. To lze realizovat třeba pomocí již v několika aplikacích použité komponenty ActiveX. Toto spojení umožní psát webové skripty v jazycích PHP, ASP a dalších, ve kterých je možné řídit, využívat a vizualizovat data ve všech klientech sítě UniNetwork.

Nasazením systému do praxe by mohlo být rovněž využití ve zmiňovaném projektu „Lablink –Virtual lab“, kde by bylo výhodné použití jeho schopností výměny dat (proměnných, souborů, konfigurací), řízení a dohledu nad různými typy systémů po Internetu.

7.2. Již realizovaná použití systému UniNetwork

Mnou navržený systém UniNetwork byl poprvé pro konkrétní aplikaci použit již během jeho vývoje. Šlo o semestrální projekt „Komunikační protokol Modbus TCP/IP“ realizovaný Ondřejem Netíkem. Úkolem tohoto projektu bylo vytvořit propojení se zařízením fy Wago, komunikujícím pomocí protokolu Modbus TCP/IP. Pan Netík naprogramoval další klientskou aplikaci systému UniNetwork, která propojuje tuto síť s řídicím automatem PLC 750-842 Programmable Fieldbus Controller od firmy Wago. Dále vytvořil druhou klientskou aplikaci, která s první aplikací komunikuje a řídí ji po UniNetwork. Příkazy pro zápis a čtení hodnot výstupů, vstupů a proměnných v PLC Wago tak může zasílat i jakýkoliv klient sítě UniNetwork.

Touto realizací se stal PLC Wago přístupný k výměně dat pro všechny ostatní klienty sítě UniNetwork (např. databázi, Matlab).

Další použití systému UniNetwork pro konkrétní účel uskutečnil Tomáš Svoboda ve své diplomové práci „Průmyslová automatizace s využitím OPC“. Zde byl systém UniNetwork použit k získávání ukázkových a simulačních databází pro server standardu OPC HDA. Konkrétně šlo o získání průběhů impulsních, přechodových a dalších charakteristik systémů simulovaných v aplikaci Matlab (více uvedeno v [10]).

8. Seznam literatury

[1] Libor Forst, Základy UNIXu a TCP/IP, http://www.ms.mff.cuni.cz/~forst/vyuka.html

[2] e-archive Jiřího Peterky, http://earchiv.isdn.cz

[3] Manuál C++ Builder - Application Developer's Guide

[4] P. Šmrha, V. Rudolf: Internetworking pomocí TCP/IP; Kopp 1994

[5] D. E. Comer, D. L. Stevens: Internetworking With TCP/IP; Prentice Hall International 1991

[6] MySQL reference manual

[7] MySQL website, http://www.mysql.com

[8] Hruška, Michael: ODBC, Softwarové noviny, číslo 12, ročník IV, prosinec 1993, str. 110,

[9] Jandos, Jaroslav: Distribuované systémy, SENZO, edice Výběr, 1. vydání, 1992

[10] Tomáš Svoboda, Průmyslová automatizace s využitím OPC, Diplomová práce, Fakulta elektrotechniky ČVUT, 2003

[11] Kateřina Pacáková, Petra Horadová, Přístup k datům, Koop 1996

[12] Microsoft Developer's Guide, http://www.msdn.microsoft.com/

[13] Matlab product documentation, Matlab help

[14] Stránky projektu Lablink, http://www.cvut.cz/fee/k317/projekty/lablink/usefullsites.html, http://dce.felk.cvut.cz/lablink/main.php

9. Přílohy

9.1. Rozhraní TCP/IP

9.1.1. Příklady užívaných portů

Číslo portu	Služba
21	FTP - File Transfer Protocol (přenos souborů)
23	telnet - Telecommunication network (interaktivní přístup ke vzdáleným počítačům)
25	SMTP - Simple Mail Transfer Protocol (přenos elektronické pošty)
80	HTTP - HyperText Transfer Protocol (přenos stránek informačního systému W W W)
119	NNTP - Network News Transfer Protocol (přenos zpráv v systému UseNet, NetNews)

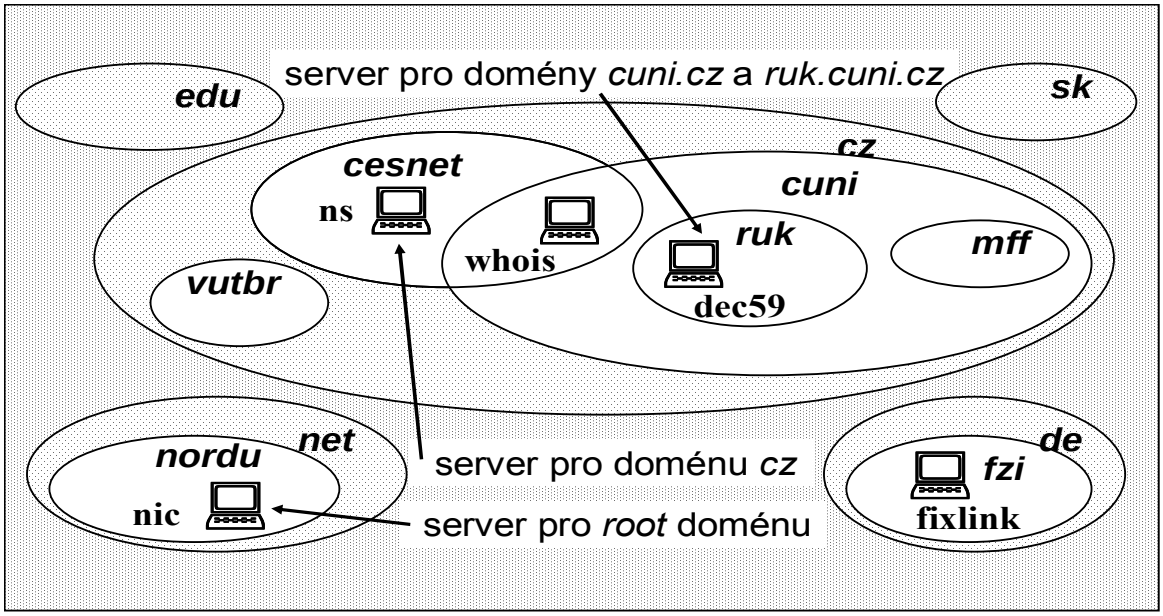
Tabulka 9.1.1: Výběr standardních „wel known“ portů

9.1.2. Struktura dat v TCP/IP

<table>
<tr><td>Version</td><td>Header Length</td><td>Service Type
(priorita)</td><td colspan="2">Total Length</td></tr>
<tr><td colspan="3">Identification
(pořadové číslo)</td><td>Flags</td><td>Fragment Offset</td></tr>
<tr><td colspan="2">Time-to-live</td><td>Protocol
(/etc/protocols)</td><td colspan="2">Header Checksum</td></tr>
<tr><td colspan="5">Source IP Address</td></tr>
<tr><td colspan="5">Destination IP Address</td></tr>
<tr><td colspan="4">Options</td><td>Padding</td></tr>
<tr><td colspan="5">Data</td></tr>
</table>

Tabulka 9.1.2: Struktura IP datagramu

9.1.3. Příklady doménových serverů



Obrázek 9.1.1: Doménový systém

9.2. Rychlost serveru MySQL, výsledky testů

Tyto testy byly prováděné za použití jedno vláknového testovacího programu. To znamená, že naměřené výsledky ukazují nejmenší možný čas. Testy byly provedeny na procesoru Pentium v prostředí OS Windows NT 4.0. Používaná cache index při testech byla 8 MB. Další výsledky testů naleznete na http://www.mysql.com/information/benchmarks.html.

Čtení 2000000 položek podle indexu	Čas v sec
mysql	367
mysql_odbc	464
db2_odbc	1206
informix_odbc	121126
ms-sql_odbc	1634
oracle_odbc	20800
solid_odbc	877
sybase_odbc	17614

Tabulka 9.2.1: Test MySQL, čtení z databáze

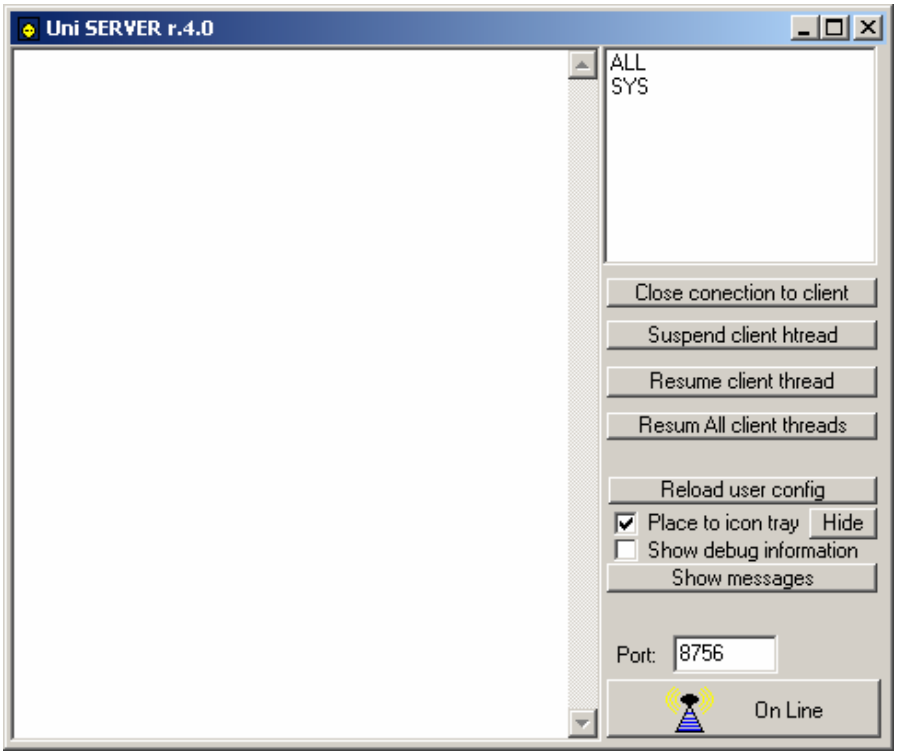
Vkládání 350768 položek	Čas v sec
mysql	381
mysql_odbc	619
db2_odbc	3460
informix_odbc	2692
ms-sql_odbc	4012
oracle_odbc	11291
solid_odbc	1801
sybase_odbc	4802

Tabulka 9.2.2: Test MySQL, ukládání do databáze

9.3. Okna a příkazy jednotlivých programů - částí systému

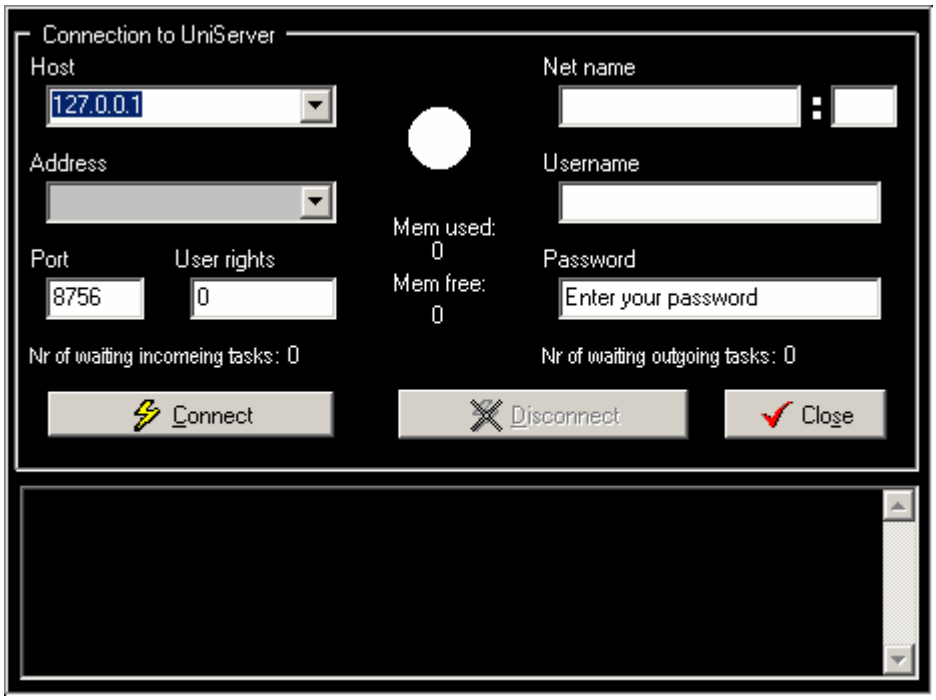
Všechny obrázky jsou zde umístěn hlavně kvůli popisu konfigurace jednotlivých programů. Na obrázky jsou uváděny reference v textu knihy.

9.3.1. UniServer

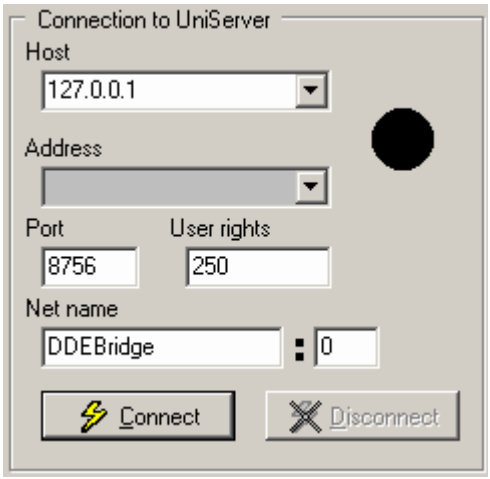


Obrázek 9.3.1: Hlavní okno UniServeru

9.3.2. Klient Builder komponenta, ActiveX klient



Obrázek 9.3.2: Formulář s nastavením obsažený v komponentě



Obrázek 9.3.3: Formulář s nastavením používaný v klientech

9.3.2.1. Příklad použití komponenty v Borland C++ Builder

Tento příklad pochází z klienta UniMessenger. Na formulář aplikace byla umístěna komponenta pro připojení k UniServeru. Komponenta byla pojmenována „UniConnectionX1“

Následující funkce je volána při stisku tlačítka pro připojení. Funkce pracuje podle algoritmu pro připojení popsaného v popisu komponenty výše v kapitole 5.4. Popis činnosti následujících funkcí se odvíjí od popisu činnosti aplikace UniMessenger v kapitole 5.17.

```
//---------------------------------------------------------------------------
void __fastcall TForm1::btnConnectClick(TObject *Sender)
{
  bool TryToConnect = true;
  pnlConnection->Enabled = false;
  btnConnect->Enabled = false;
  UniConnectionX1->NetName = edNetName->Text + ":" + IntToStr(edNrConn-
>Text.ToInt());
  UniConnectionX1->UserName = "Messenger";
  UniConnectionX1->Password = "uni";
  UniConnectionX1->RequestedRights = 250;
  UniConnectionX1->DesAddress = "SYS";
  UniConnectionX1->Host = cbMachine->Text;
  UniConnectionX1->HostName = cbAddress->Text;
  UniConnectionX1->Port = edPort->Text.ToInt();
  while (TryToConnect) {
    try {
      UniConnectionX1->NetName = edNetName->Text + ":" + IntToStr(edNrConn-
>Text.ToInt());
      UniConnectionX1->Connect();
      TryToConnect = false;
    } catch (const Exception &E) {
      if (E.Message == "Bad LogIn! - Addreess already exists - UserNotLogged") {
        int i = edNrConn->Text.ToInt();
        edNrConn->Text = ++i;
        Sleep(1000);
        Repaint();
      } else {
        TryToConnect = false;
        MessageDlg("Error:" + E.Message, mtError, TMsgDlgButtons() << mbOK, 0);
        pnlConnection->Enabled = true;
        btnConnect->Enabled = true;
        return;
      }
    }
  }
  btnDisconnect->Enabled = true;
  ConnectionRun = true;
  TrayMessage(NIM_MODIFY);
}
//---------------------------------------------------------------------------
```

Dále uvedená část kódu je funkce, která je volána pokud komponenta obdrží od serveru jednu nebo více úloh (událost komponenty „OnWaitingResp“). Jde o smyčku, která zpracovává úlohy (příkazy) dokud nějaké jsou ve frontě příchozích zpráv.

```
//---------------------------------------------------------------------------
void __fastcall TForm1::UniConnectionX1WaitingResp(TObject *Sender)
{
  bool CmdExecuted = false;
```

```
  while ((UniConnectionX1->NrOfWaitingIncomTask > 0) && (UniConnectionX1-
>CommInProgress)) {
    CmdExecuted = false;
    UniConnectionX1->RecieveTask();
    AnsiString Command = UniConnectionX1->Command; Command = Command.UpperCase();
//.............................................................................
    if (!CmdExecuted && (Command == "GETVAR")) {
      CmdExecuted = true;
    }
//.............................................................................
    if (!CmdExecuted && (Command == "PUTVAR")) {
      CmdExecuted = true;
    }
//.............................................................................
    if (!CmdExecuted && (Command == "GETCLIENTLIST")) {
      CmdExecuted = true;
    }
//.............................................................................
    if (!CmdExecuted && (Command == "VARSAVED")) {
      CmdExecuted = true;
    }
//.............................................................................
    if (!CmdExecuted && (Command == "ERRORMESSAGE")) {
      AnsiString Text;
      Text += "-------------------------------------------------------------\r\n";
      Text += "ERRORMESSAGE\r\n";
      Text +=
".............................................................................
....................\r\n";
      Text += UniConnectionX1->ReadDatString();
      Text += "\r\nFrom addr: " + UniConnectionX1->SrcAddress;
      Text += "\r\nID: " + IntToStr(UniConnectionX1->TaskID);
      Text += "\r\nRecieved: " + FormatDateTime("(d.m.y | h:nn:ss.z)", Now());
      Text += "\r\n-----------------------------------------------------";
      memMessages->Lines->Insert(0, Text);
      memMessages->SelStart = 0;
      memMessages->SelLength = Text.Length();
      memMessages->SelAttributes->Color = clRed;
      if (cbShowInformationDlg->Checked) MessageDlg (Text, mtError, TMsgDlgButtons()
<< mbOK, 0);
      CmdExecuted = true;
      Show1Click(Sender);
      Application->BringToFront();
    }
//.............................................................................
    if (!CmdExecuted && (Command == "INFOMESSAGE")) {
      AnsiString Text;
      Text += "-------------------------------------------------------------\r\n";
      Text += "INFOMESSAGE\r\n";
      Text +=
".............................................................................
....................\r\n";
      Text += UniConnectionX1->ReadDatString();
      Text += "\r\nFrom addr: " + UniConnectionX1->SrcAddress;
      Text += "\r\nID: " + IntToStr(UniConnectionX1->TaskID);
      Text += "\r\nRecieved: " + FormatDateTime("(d.m.y | h:nn:ss.z)", Now());
      Text += "\r\n-----------------------------------------------------";
      memMessages->Lines->Insert(0, Text);
      memMessages->SelStart = 0;
      memMessages->SelLength = Text.Length();
      memMessages->SelAttributes->Color = clYellow;
      if (cbShowInformationDlg->Checked) MessageDlg (Text, mtInformation,
TMsgDlgButtons() << mbOK, 0);
      CmdExecuted = true;
      Show1Click(Sender);
```

```
      Application->BringToFront();
    }
//.............................................................................
    if (!CmdExecuted && (Command == "YESNOMESSAGE")) {
      AnsiString Text;
      Text += "-------------------------------------------\r\n";
      Text += "YESNOMESSAGE\r\n";
      Text +=
".............................................................................
...................\r\n";
      Text += UniConnectionX1->ReadDatString();
      Text += "\r\nFrom addr: " + UniConnectionX1->SrcAddress;
      Text += "\r\nID: " + IntToStr(UniConnectionX1->TaskID);
      Text += "\r\nRecieved: " + FormatDateTime("(d.m.y | h:nn:ss.z)", Now());
      Text += "\r\n-------------------------------------------------------------";
      memMessages->Lines->Insert(0, Text);
      memMessages->SelStart = 0;
      memMessages->SelLength = Text.Length();
      memMessages->SelAttributes->Color = clWhite;
      UniConnectionX1->SwitchAddress();
      UniConnectionX1->Command = "YESNOANSWER";
      if (MessageDlg(Text, mtConfirmation, TMsgDlgButtons() << mbYes <<mbNo, 0) ==
mrYes)
        UniConnectionX1->WriteDatBool(true);
      else UniConnectionX1->WriteDatBool(false);
      UniConnectionX1->SendTask();
      CmdExecuted = true;
      Show1Click(Sender);
      Application->BringToFront();
    }
//.............................................................................
    if (!CmdExecuted) ShowMessage("Recieved unknown command:" + Command + "!");
  }
}
//-----------------------------------------------------------------------------
```

Další programové konstrukce pro práci s komponentou a celkově pro přístup do UniNetwork jsou přehledně a snadno k prostudování použity ve zdrojových kódech aplikací systému UniNetwork.

9.3.2.2. Příklad použití ActiveX objektu komponenty v jazyce Visual Basic

Příklad je ze sešitu aplikace MS Excel. Na plochu sešitu byl umístěn vizuální prvek ActiveX pro připojení do UniNetwork. Kód je podobný kódu v jazyce C++. Příklad umožňuje připojit komponentu ze sešitu Excel do sítě, zasílat úlohy pro ostatní klienty a zpracovávat úlohy přijaté komponentou od ostatních klientů.

Funkce volaná pro připojení. Tentokrát jednodušší varianta, která je bez inkrementace pořadového čísla klienta.

```
Private Sub btnConnect_Click()
  UniCon.UserName = "Uni"
  UniCon.Password = "client"
  UniCon.Host = Cells(2, 2)
  UniCon.NetName = Cells(3, 2)
  UniCon.Connect

  UniCon.ResetTask
  UniCon.SrcAddress = ""
  UniCon.DesAddress = "SYS"
```

```
  UniCon.Command = "GETCLIENTLIST"
  UniCon.SendTask

End Sub
```

Funkce pro zaslání požadavku o proměnnou – úloha „GETVAR“

```
Private Sub btnGetVar_Click()
 UniCon.ResetTask
 UniCon.SrcAddress = ""
 UniCon.DesAddress = Cells(23, 10)
 UniCon.Command = "GETVAR"
 UniCon.WriteDatString (Cells(24, 10))
 UniCon.SendTask
End Sub
```

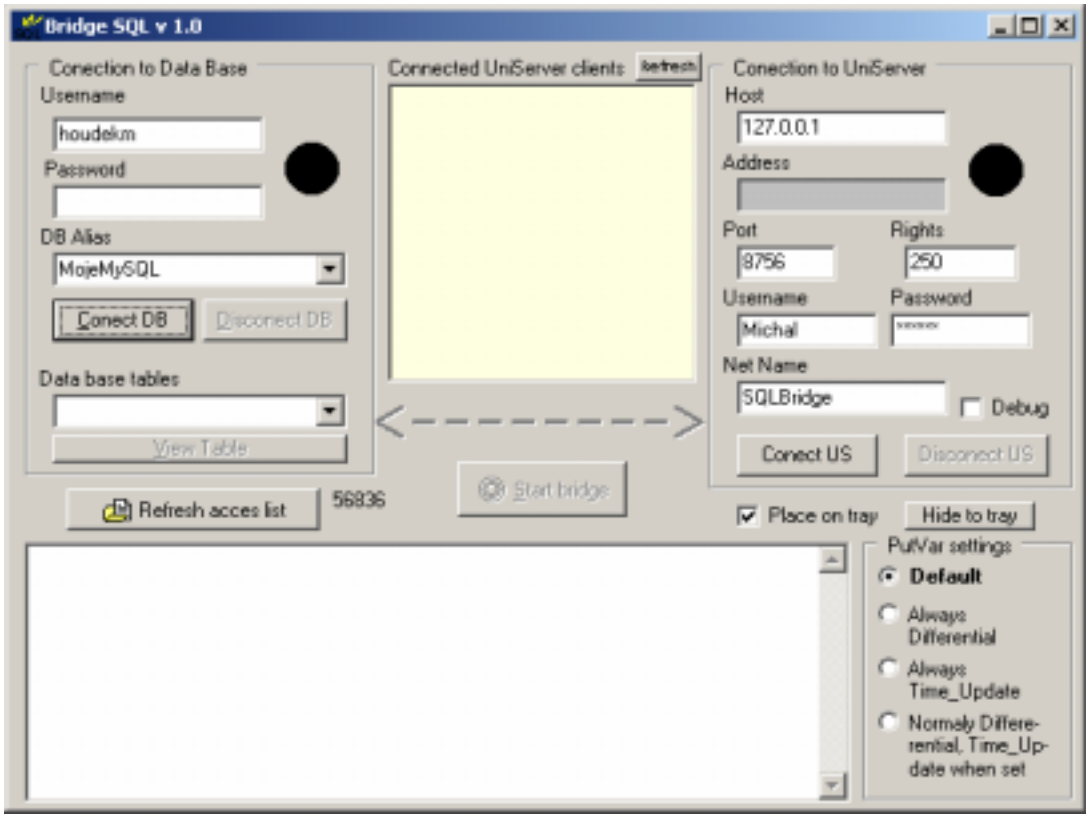
A opět příklad funkce reagující na příchozí úlohy.

```
Private Sub UniCon_OnWaitingResp()
  Dim Command As String
  Dim CmdExecuted As Boolean
  Dim VarName As String
  Dim VarType As String
  Dim VarTime As String
  Dim iRow, iCol As Integer
  Dim TmpA, TmpB As Variant
  While UniCon.NrOfWaitingIncomTask > 0
    CmdExecuted = False
    UniCon.RecieveTask
    Command = UCase(UniCon.Command)
    Cells(7, 2) = Command
    Cells(9, 2) = UniCon.SrcAddress
'-----------------------------------------------------------------------
 If Command = "GETVAR" Then
'  VarName =
  Cells(1000, 1) = UniCon.ReadDatString
  TmpA = InStr(1, Cells(1000, 1), "\")
  If TmpA <> 0 Then
    Cells(1000, 1) = Mid(Cells(1000, 1), TmpA + 1)
  End If
  TmpA = InStr(1, Cells(1000, 1), "R")
  TmpB = InStr(1, Cells(1000, 1), "C")
  iRow = Mid(Cells(1000, 1), TmpA + 1, TmpB - 2)
  iCol = Mid(Cells(1000, 1), TmpB + 1)
  UniCon.WriteDatString (UniCon.NetName + "\" + Cells(1000, 1))
  UniCon.AddDatString ("STR")
  UniCon.AddDatString (UniCon.ActualTime)
  UniCon.AddDatString (Cells(iRow, iCol))
  UniCon.Command = "PUTVAR"
  UniCon.SwitchAddress
  UniCon.SendTask
  CmdExecuted = True
 End If
'-----------------------------------------------------------------------
 If Command = "PUTVAR" Then
'  VarName =
  Cells(1000, 1) = UniCon.ReadDatString
'  VarType = UniCon.ReadDatStrin
  Cells(1000, 2) = UniCon.ReadDatString
'  VarTime = UniCon.ReadDatString
  Cells(1000, 3) = UniCon.ReadDatString
  TmpA = InStr(1, Cells(1000, 1), "\")
  If TmpA <> 0 Then
    Cells(1000, 1) = Mid(Cells(1000, 1), TmpA + 1)
  End If
  On Error GoTo ErrorHendl
```

```
    TmpA = InStr(1, Cells(1000, 1), "R")
    TmpB = InStr(1, Cells(1000, 1), "C")
    iRow = Mid(Cells(1000, 1), TmpA + 1, TmpB - 2)
    iCol = Mid(Cells(1000, 1), TmpB + 1)
    GoTo NormHandle
ErrorHendl:
    iRow = 25
    iCol = 10
NormHandle:
  Cells(8, 2) = Cells(1000, 1) + " " + Cells(1000, 3)
  Dim aaa As String
  aaa = Cells(1000, 2)
  If aaa = "INT" Then
    Cells(iRow, iCol) = UniCon.ReadDatLong
  End If
  If aaa = "FLOAT" Then
    Cells(iRow, iCol) = UniCon.ReadDatSingle
  End If
  If aaa = "STR" Then
    Cells(iRow, iCol) = UniCon.ReadDatString
  End If
  CmdExecuted = True
 End If
'-----------------------------------------------------------------------
 If Command = "TIMEVAR" Then
  Dim tv As Variant
  Dim ind
  Dim max
  Dim max2
  tv = UniCon.ReadTimeVar
  max = UBound(tv, 1)
  max2 = UBound(tv, 1)
  If max > 20 Then
    max = 20
  End If
  If max > max2 - 1 Then
    max = max2 - 1
  End If
  For ind = max To 1 Step -1
    Cells(ind + 14, 1) = tv(max2 - ind, 1)
    Cells(ind + 14, 2) = tv(max2 - ind, 2)
  Next ind
  CmdExecuted = True
 End If
'-----------------------------------------------------------------------
 If Command = "GETCLIENTLIST" Then
  ListBox1.List = UniCon.ReadDatStringToVariantArray
  CmdExecuted = True
 End If
'-----------------------------------------------------------------------
 If Command = "ERRORMESSAGE" Then
  Cells(8, 2) = UniCon.ReadDatStringToVariantArray
  CmdExecuted = True
 End If
'-----------------------------------------------------------------------
 If Not CmdExecuted Then
  Cells(8, 2) = "Unknown command"
 End If
'-----------------------------------------------------------------------
 Wend
End Sub
```

Celý zdrojový kód je umístěn na přiloženém CD v adresáři Examples.

9.3.3. SQL Bridge



Obrázek 9.3.4: Hlavní okno SQL Bridge

```
CREATE TABLE VarDef
(
  ID        INTEGER UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE,
  Name            CHAR (40) NOT NULL UNIQUE,
  Type            CHAR (6) NOT NULL,
  ParentID        INTEGER UNSIGNED ZEROFILL NOT NULL,
  Info            VARCHAR (255),
  PRIMARY KEY     (ID)
)

CREATE TABLE VarStr
(
  ID        INTEGER UNSIGNED NOT NULL,
  ReadTime DATETIME NOT NULL,
  ReadTimeMSec    INTEGER ZEROFILL NOT NULL,
  Value           VARCHAR(255),
  PRIMARY KEY     (ReadTime, ReadTimeMSec, ID)
)

CREATE TABLE VarLong
(
  ID        INTEGER UNSIGNED NOT NULL,
```

```
    ReadTime DATETIME NOT NULL,
    ReadTimeMSec  INTEGER ZEROFILL NOT NULL,
    Value         INTEGER,
    PRIMARY KEY   (ReadTime, ReadTimeMSec, ID)
  )

  CREATE TABLE VarFloat
  (
    ID       INTEGER UNSIGNED NOT NULL,
    ReadTime DATETIME NOT NULL,
    ReadTimeMSec  INTEGER ZEROFILL NOT NULL,
    Value         FLOAT,
    PRIMARY KEY   (ReadTime, ReadTimeMSec, ID)
  )
```

Tabulka 9.3.1: Skript pro definici tabulek SQL Bridge

9.3.3.1. Podporované příkazy

EXESQLTXT

Popis: příkaz na provedení SQL dotazu, výsledek se vrací v textové formě tabulky
Data příkazu: AnsiString SQLQuery
Význam dat: SQLQuery – SQL dotaz
Odpověď: EXESQLTXT
Data odpovědi: AnsiString Txt
Význam dat: Txt – textový formát tabulky, řádky odděleny tabelátory

EXESQLDAT

Popis: příkaz na provedení SQL dotazu, výsledek se vrací ve formě struktury dat, reprezentující tabulku
Data příkazu: AnsiString SQLQuery
Význam dat: SQLQuery – SQL dotaz
Odpověď: EXESQLDAT
Data odpovědi: long RowCount – počet řádků v tabulce
long FieldCount – počet sloupců v tabulce
AnsiString FieldType – typ sloupce, položka se opakuje tolikrát kolik je sklopců
AnsiString FieldName – jméno sloupce, položka se opakuje tolikrát kolik je sklopců
dále jsou přidávány jednotlivé buňky tabulky vždy celý řádek zprava doleva a následuje další řádek. Typ přidávaných dat odpovídá typu sloupce v databázi.
Význam dat: tabulka výsledku SQL dotazu

SQLNORESULT

Popis: je to odpověď na příkaz EXESQL… v případě, že odpověď na tento dotaz je prázdná.
Data příkazu: AnsiString SQLQuery
Význam dat: SQLQuery – SQL dotaz

GETVARSTRUCTXT

Popis: příkaz ke zjištění struktury závislostí mezi datovými položkami v databázi a zjištění jejich typu
Data příkazu: -
Význam dat: -
Odpověď: VARSTRUCTXT
Data odpovědi: AnsiString Txt
Význam dat: Txt – textový formát tabulky vyjadřující strukturu datových položek v databázi

GETVARSTRUCDAT

Popis: příkaz ke zjištění struktury závislostí mezi datovými položkami v databázi a zjištění jejich typu, výsledek se vrací ve formě tabulky
Data příkazu: -
Význam dat: -
Odpověď: VARSTRUCDAT
Data odpovědi: struktura dat v této úloze je stejná jako u EXESQLDAT
Význam dat: tabulka vyjadřující strukturu datových položek v databázi

GETVARLISTTXT

Popis: příkaz ke zjištění jmen všech definovaných proměnných a jejich typu
Data příkazu: -
Význam dat: -
Odpověď: VARLISTTXT
Data odpovědi: AnsiString Txt
Význam dat: Txt – textový formát tabulky

GETVARLISTDAT

Popis: příkaz ke zjištění jmen všech definovaných proměnných a jejich typu
Data příkazu: -
Význam dat: -
Odpověď: VARLISTDAT
Data odpovědi: long RowCount – počet řádků v tabulce
AnsiString ItemName – jméno proměnné
AnsiString ItemType – typ proměnné

Předchozí dvě položky se opakují tolikrát kolik je definovaných proměnných v databázi.
Význam dat: tabulka jmen a typů proměnných

PUTVAR

Popis: uložení proměnné do databáze
Data příkazu: AnsiString VarName
AnsiString VarType
AnsiString TimeStamp
Data
AnsiString SaveType – nepovinný parametr
Význam dat: VarName – jméno proměnné
VarType – typ proměnné {INT, FLOAT, STR}
TimeStamp – čas získání proměnné
Data – hodnota proměnné, typ podle VarType
SaveType – dodatečné informace pro uložení proměnné do databáze. {DIFFERENTIONAL, TIME_UPDATE}
Odpověď: VARSAVED
Data odpovědi: AnsiString VarName
Význam dat: VarName – jméno proměnné

GETVAR

Popis: čtení poslední hodnoty proměnné v databázi
Data příkazu: AnsiString VarName
AnsiString BackVarName.
Význam dat: VarName – jméno proměnné
BackVarName – jméno proměnné uvedené v odpovědi – nepovinný parametr
Odpověď: PUTVAR

GETVARAPROX

Popis: čtení hodnoty proměnné v databázi, která je časově nejbližší zadanému času
Data příkazu: AnsiString VarName
AnsiString VarTime
AnsiString BackVarName.
Význam dat: VarName – jméno proměnné
VarTime – čas kdy chceme zjistit hodnotu proměnné
BackVarName – jméno proměnné uvedené v odpovědi – nepovinný parametr
Odpověď: PUTVAR

GETTIMEVAR

Popis: čtení všech hodnot proměnné v databázi, které leží mezi zadanými časy
Data příkazu: AnsiString VarName

AnsiString TimeFrom
AnsiString TimeTo
Význam dat: VarName – jméno proměnné
TimeFrom – čas od kdy chceme zjistit hodnotu proměnné
TimeTo – čas do kdy chceme zjistit hodnotu proměnné
Odpověď: TIMEVAR
Data odpovědi: long RowCount
AnsiString VarType
AnsiString VarTime
VarValue
Předchozí dvě položky se opakují tolikrát kolik je záznamů v databázi
Význam dat: RowCount – počet načtených hodnot
VarType – typ proměnné
VarTime – čas získání hodnoty proměnné
VarValue – hodnota proměnné, typ závisí na VarType.

CLEARVAR

Popis: smazání hodnot proměnné z databáze v určitém časovém rozsahu
Data příkazu: AnsiString VarName
AnsiString TimeFrom
AnsiString TimeTo
Význam dat: VarName – jméno proměnné
TimeFrom – čas od kdy chceme hodnoty proměnné vymazat z databáze
TimeTo – čas do kdy chceme hodnoty proměnné vymazat z databáze
Odpověď: CLEARVAR_OK
Data odpovědi: AnsiString Result
Význam dat: Result – vždy „OK“, v případě neúspěchu se vrací jako odpověď ERRORMESSAGE

ADDVAR

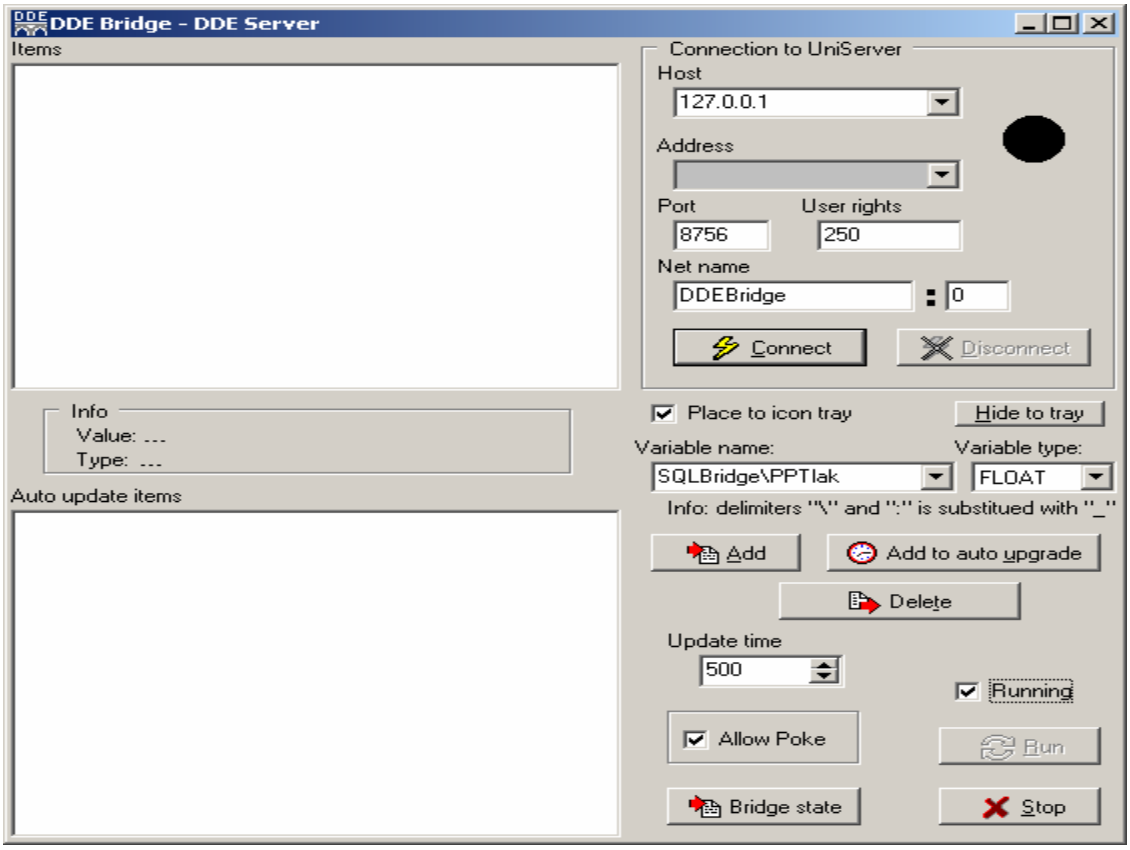
Popis: přidání nové proměnné do databáze
Data příkazu: AnsiString VarName
AnsiString VarType
long ParentID
AnsiString VarInfo
Význam dat: VarName – jméno proměnné
VarType – typ proměnné
ParentID – identifikace nadřazené proměnné.
VarInfo – slovní popis proměnné
Odpověď: ADDVAR_OK
Data odpovědi: AnsiString Result

Význam dat: Result – vždy „OK“, v případě neúspěchu se vrací jako odpověď ERRORMESSAGE

DELVAR

Popis: smazání proměnné z databáze
Data příkazu: AnsiString VarName
Význam dat: VarName – jméno proměnné
Odpověď: DELVAR_OK
Data odpovědi: AnsiString Result
Význam dat: Result – vždy „OK“, v případě neúspěchu se vrací jako odpověď ERRORMESSAGE

9.3.4. DDE Bridge



Obrázek 9.3.5: Hlavní okno DDE Bridge

9.3.4.1. Podporované příkazy

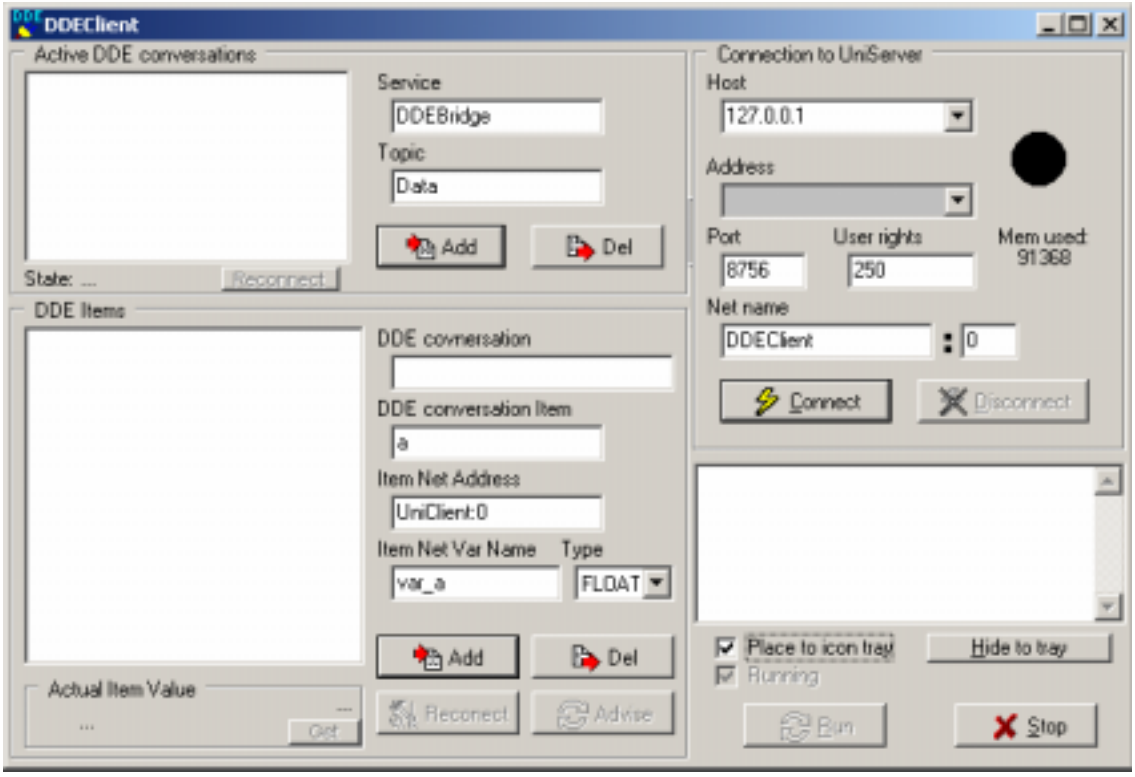
GETVAR

Příkaz je stejný jako u SQL Bridge

PUTVAR

Příkaz je stejný jako u SQL Bridge

9.3.5. DDE Klient



Obrázek 9.3.6: Hlavní okno DDE Client

9.3.5.1. Podporované příkazy

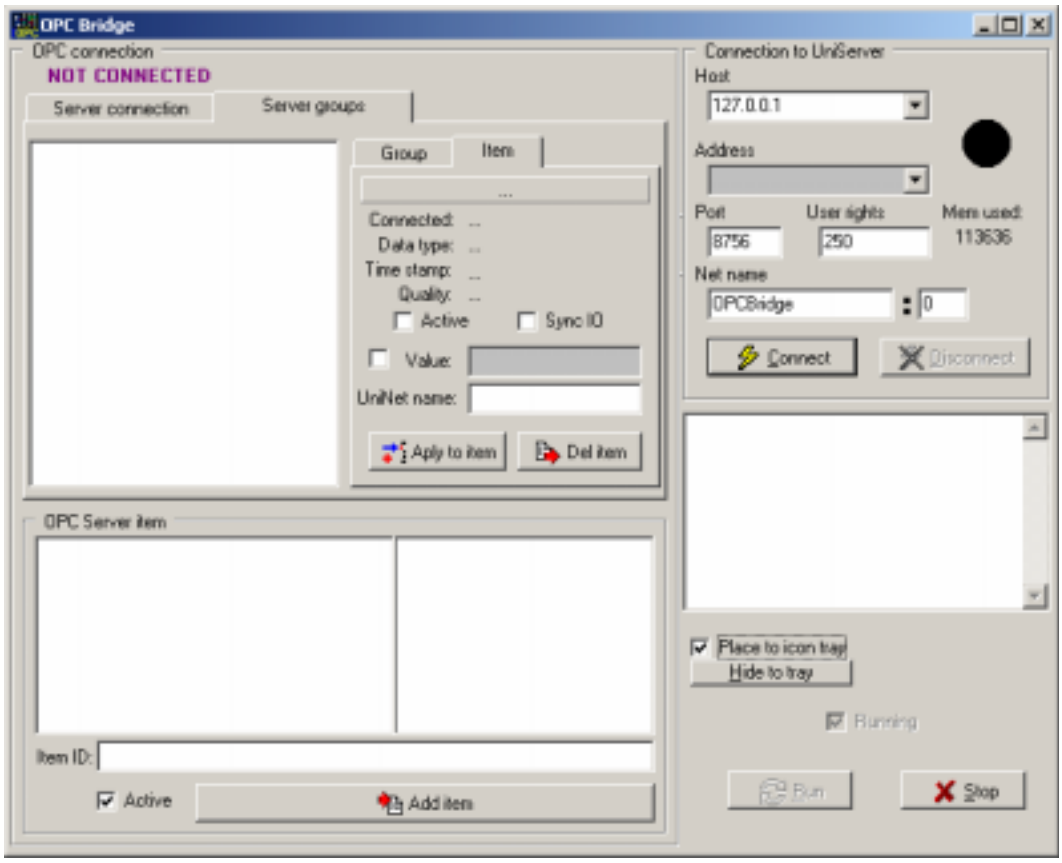
GETVAR

Příkaz je stejný jako u SQL Bridge

PUTVAR

Příkaz je stejný jako u SQL Bridge

9.3.6. OPC Bridge



Obrázek 9.3.7: Hlavní okno OPC Bridge

9.3.6.1. Podporované příkazy

GETVAR

Příkaz je stejný jako u SQL Bridge

PUTVAR

Příkaz je stejný jako u SQL Bridge

GETOPCGROUPSTRUC

Popis: příkaz ke zjištění struktury nadefinovaných skupin na OPC serveru
Data příkazu: -
Význam dat: -
Odpověď: OPCGROUPSTRUC
Data odpovědi: AnsiString Txt
Význam dat: Txt – seznam skupin OPC

GETOPCVARSTRUC

Popis: příkaz ke zjištění struktury proměnných na OPC serveru
Data příkazu: -
Význam dat: -
Odpověď: OPCVARSTRUC
Data odpovědi: AnsiString Txt
data se opakují pro každou úroveň stromu hierarchie stromu proměnných na OPC serveru
Význam dat: Txt – seznam skupin OPC na jedné úrovni stromu

GETOPCGROUPINFO

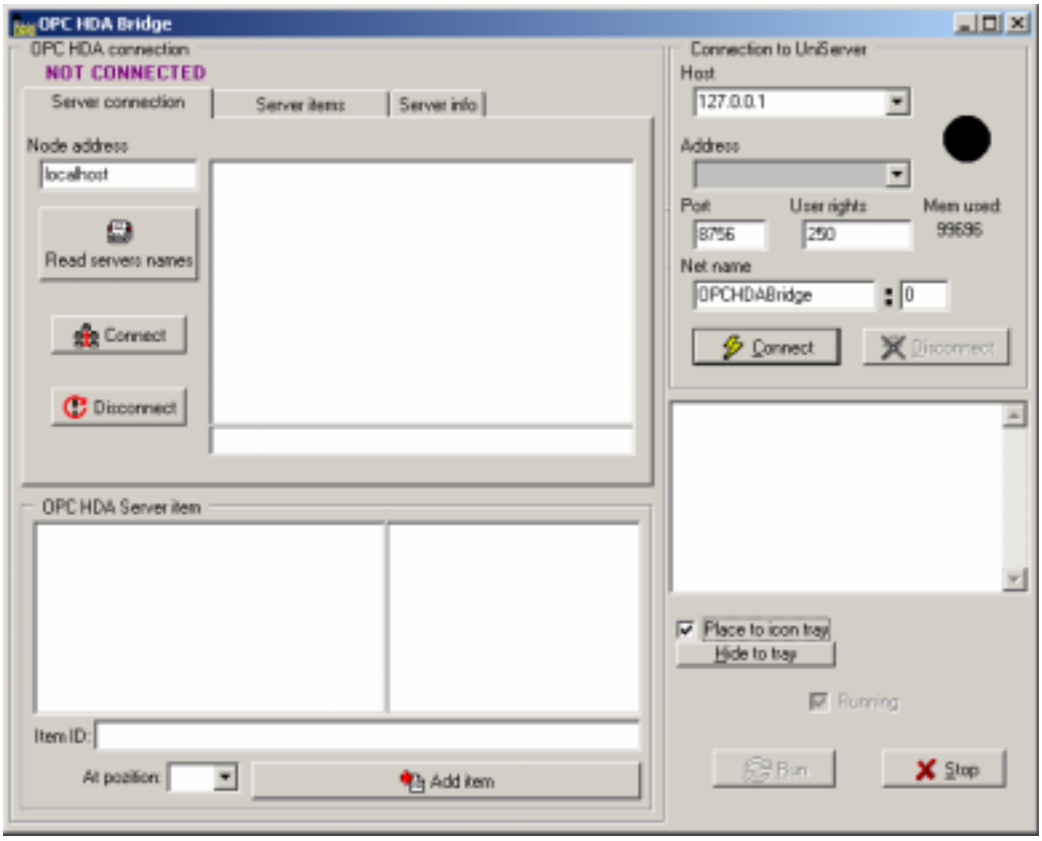
Popis: zjištění informací o OPC skupině
Data příkazu: AnsiString GroupName
Význam dat: GroupName – jméno skupiny
Odpověď: OPCGROUPNFO
Data odpovědi: AnsiString Name
bool Active
long UpTime
. single DeadBand
Význam dat: Name – jméno skupiny
Active – zda je skupina na serveru aktivní
UpTime – update time skupiny
DeadBand – pásmo necitlivosti proměnných ve skupině na změny hodnoty

GETOPCITEMINFO

Popis: zjištění informací o OPC položce
Data příkazu: AnsiString Name
AnsiString GroupName
AnsiString FullName
Význam dat: Name – jméno OPC položky
GroupName – jméno skupiny
FullName – kompletní jméno
Odpověď: OPCITEMINFO
Data odpovědi: AnsiString Name
AnsiString Nick
bool Connected
AnsiString DataType
AnsiString TimeStamp
long Quality
bool SyncIO
bool Active
AnsiString Value
Význam dat: Name – jméno položky
Nick – jméno položky pro UniNetwork
Connected – zda je připojeno zařízení na OPC server
TimeStamp – čas načtení položky
Quality – důvěryhodnost položky

SyncIO – synchronní či asynchronní čtení
Value – hodnota proměnné

9.3.7. OPC HDA Bridge



Obrázek 9.3.8: Hlavní okno OPC HDA Bridge

9.3.7.1. Podporované příkazy

GETOPCHDAVARSTRUC

Popis: příkaz ke zjištění struktury proměnných na OPC HDA serveru
Data příkazu: -
Význam dat: -
Odpověď: OPCHDAVARSTRUC
Data odpovědi: AnsiString Txt
data se opakují pro každou úroveň stromu hierarchie stromu proměnných na OPC HDA serveru
Význam dat: Txt – seznam skupin OPC na jedné úrovni stromu

GETOPCHDAITEMS

Popis: zjištění nakonfigurovaných položek na serveru
Data příkazu: -
Význam dat: -
Odpověď: OPCHDAITEMS
Data odpovědi: AnsiString Txt
Význam dat: Txt – seznam položek serveru

GETOPCHDAITEMINFO

Popis: získání informací o položce na serveru
Data příkazu: -
Význam dat: -
Odpověď: OPCHDAITEMINFO
Data odpovědi: AnsiString AttribList
AnsiString AttribDesc
AnsiString Attribs
Význam dat: AttribList – seznam atributů položky
AttribDesc – Slovní popis významu atributů
Attribs – hodnoty atributů

GETOPCHDAAGGREG

Popis: získání informací o agregačních funkcích
Data příkazu: -
Význam dat: -
Odpověď: OPCHDAAGGREG
Data odpovědi: AnsiString AggList
AnsiString AttribDesc
Význam dat: AttribList – seznam agregačních funkcí
AttribDesc – Slovní popis funkcí

GETRAWDATA

Popis: čtení neupravených dat přímo z databáze
Data příkazu: AnsiString ItemName
AnsiString TimeFrom
AnsiString TimeTo
long MaxItems
Význam dat: ItemName – jméno položky
TimeFrom – počátek čtení dat.
TimeTo – konec čtení dat.
MaxItems – maximální počet položek
Odpověď: RAWDATA
Data odpovědi: AnsiString ItemName
long ItemIndex
AnsiString TimeFrom
AnsiString TimeTo
long MaxItems
long ItemsCount

AnsiString Value
AnsiString TimeStamp
AnsiString Qualities
Poslední tři položky se opakují tolikrát kolik bylo načteno dat z databáze (ItemsCount),

Význam dat: ItemIndex – pořadové číslo položky na serveru
ItemsCount – počet načtených položek
Value – hodnota položky
TimeStamp – čas odečtu položky
Qualities – důvěryhodnost položky

GETPROCESSEDDATA

Popis: čtení dat převzorkovaných po nějakém časovém intervalu a zpracovaných agregační funkcí

Data příkazu: AnsiString ItemName
AnsiString TimeFrom
AnsiString TimeTo
AnsiString ResampleTime
long AggIndex

Význam dat: ItemName – jméno položky
TimeFrom – počátek čtení dat.
TimeTo – konec čtení dat.
ResampleTime – časový interval převzorkování
AggIndex – index použité agregační funkce

Odpověď: PROCESSEDDATA

Data odpovědi: struktura odpovědi je stejná jako u GETRAWDATA

GETDATAATTIME

Popis: čtení dat z databáze získaných ve stanovených časech

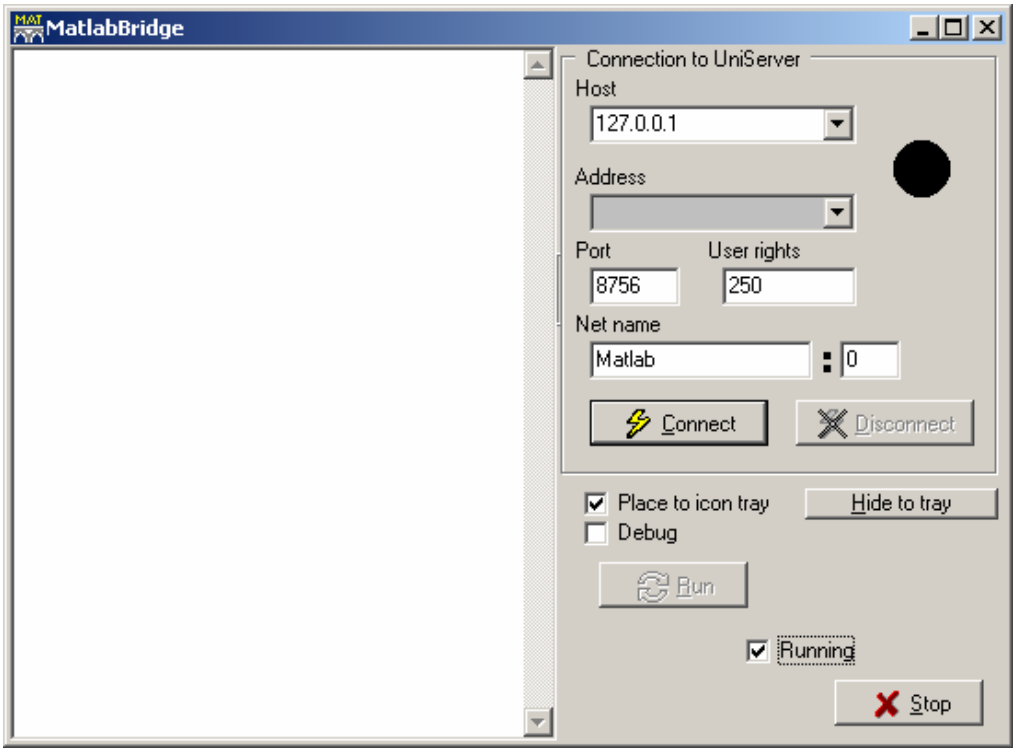
Data příkazu: AnsiString ItemName
AnsiString ListTimes

Význam dat: ItemName – jméno položky
ListTimes – seznam času ve kterých chceme číst hodnoty položek

Odpověď: PROCESSEDDATA

Data odpovědi: struktura odpovědi je stejná jako u GETRAWDATA

9.3.8. Matlab Bridge



Obrázek 9.3.9: Hlavní okno Matlab Bridge

9.3.8.1. Podporované příkazy

GETVAR

Příkaz je stejný jako u SQL Bridge

PUTVAR

Příkaz je stejný jako u SQL Bridge

MATLABDOCOMMAND

Popis: spuštění příkazu v Matlabu
Data příkazu: AnsiString Cmd
Význam dat: Cmd - příkaz
Odpověď: MATLABDOCOMMAND
Data odpovědi: AnsiString Resp
Význam dat: Resp – odpověď Matlabu

LOCKCLIENT

Popis: zamknutí Matlabu pro výhradní užívání klientem, který příkaz zaslal
Data příkazu: -
Význam dat: -

Odpověď: CLIENTLOCKED nebo CLIENTINUSE
Data odpovědi: -
Význam dat: -

UNLOCKCLIENT

Popis: odemknutí Matlabu pro užívání všemi klienty
Data příkazu: -
Význam dat: -
Odpověď: CLIENTUNLOCKED
Data odpovědi: -
Význam dat: -

MATLABGETARRAY

Popis: načtení matice z Matlabu
Data příkazu: AnsiString Name
Význam dat: Name – jméno matice
Odpověď: MATLABGETARRAY
Data odpovědi: long Rows
long .Cols
RealPart
ImagPart
Význam dat: Rows- počet řádků matice
Cols – počet sloupců matice
RealPart – reálná část matice, uložená po sloupcích, buňky jsou formátu double
ImagPart – imaginární část matice, uložená po sloupcích, buňky jsou formátu double

MATLABGETARRAYIN

Popis: varianta předchozího pro matice s prvky integer

MATLABGETARRAYCH

Popis: varianta předchozího pro matice s prvky char a řetězce

MATLABPUTARRAY

Popis: uložení matice do Matlabu
Data příkazu: AnsiString Name
long Rows
long .Cols
RealPart
ImagPart
Význam dat: Name – jméno matice
Rows- počet řádků matice
Cols – počet sloupců matice
RealPart – reálná část matice, uložená po sloupcích, buňky jsou formátu double
ImagPart – imaginární část matice, uložená po sloupcích, buňky jsou formátu double

Odpověď: MATLABGETARRAY
Data odpovědi: AnsiString Name
Význam dat: Name – potvrzení, že matice byla uložena

MATLABPUTARRAYIN

Popis: varianta předchozího pro matice s prvky integer

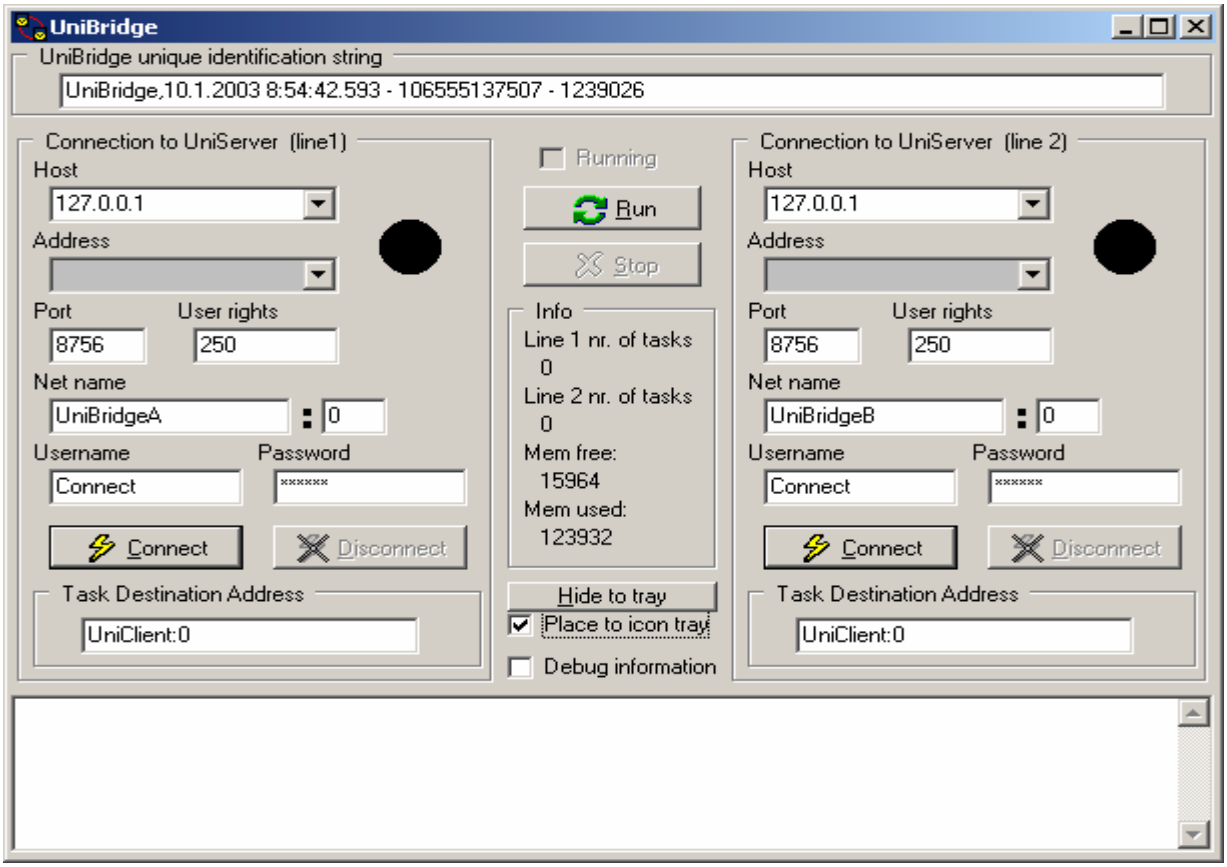
MATLABPUTARRAYCH

Popis: varianta předchozího pro matice s prvky char a řetězce

MATLABGETPICTURE

Popis: příkaz pro načtení matice reprezentující obrázek v Matlabu. Při vykonávání příkazu je určen ty obrázku (BW/RGB) a podle toho vybrána použitá struktura dat odpovědi. Pro bližší informace odkazuji do zdrojového kódu v souboru „MatlabBridge\Main.cpp“ řádka 677.

9.3.9. UniBridge



Obrázek 9.3.10: Hlavní okno programu UniBridge

9.3.9.1. Podporované příkazy

UNIBRIDGECOMMAND

Popis: skupina příkazů pro čtení nebo zápis konfigurace propojených sítí
Data příkazu: AnsiString BridgeID
AnsiString BridgeCmd
Data
Význam dat: BridgeID – string identifikující cílový UniBridge
BridgeCmd – příkaz pro UniBridge
Data – další data pro příkaz
Odpověď: UNIBRIDGESTAT
Data odpovědi: AnsiString BridgeID
AnsiString BridgeStat
Data
Význam dat: BridgeStat – identifikace odpovědi z UniBridge

9.3.9.2. Podporované příkazy typu UNIBRIDGECOMMAND

SETBRIDGEIDSTRING

Popis: nastavení stringu identifikujícího UniBridge
Data příkazu: AnsiString BridgeID
Význam dat: BridgeID – string identifikující cílový UniBridge
Odpověď: -

SETTASKDESADDR

Popis: nastavení adresy v lokální síti, na kterou budou chodit pakety ze vzdálené sítě
Data příkazu: AnsiString Addr
Význam dat: Addr – adresa v síti příkaz zasílajícího klienta
Odpověď: -

SETREMOTETASKDESADDR

Popis: nastavení adresy ve vzdálené síti, na kterou budou chodit pakety z lokální sítě
Data příkazu: AnsiString Addr
Význam dat: Addr – adresa v připojené síti
Odpověď: -

GETBRIDGEIDSTRING

Popis: příkaz ke zjištění ID daného UniBridge
Data příkazu: -
Odpověď: BRIDGEIDSTRING
Data odpovědi: AnsiString BridgeID
Význam dat: BridgeID – string identifikující UniBridge

GETTASKDESADDR

Popis: příkaz ke zjištění adresy v lokální síti, na chodí pakety ze vzdálené sítě
Data příkazu: -
Odpověď: TASKDESADDR
Data odpovědi: AnsiString Addr
Význam dat: Addr – adresa v síti příkaz zasílajícího klienta

GETREMOTETASKDESADDR

Popis: příkaz ke zjištění adresy ve vzdálené síti, na kterou budou chodit pakety z lokální sítě
Data příkazu: -
Odpověď: REMOTETASKDESADDR
Data odpovědi: AnsiString Addr
Význam dat: Addr – adresa v připojené síti

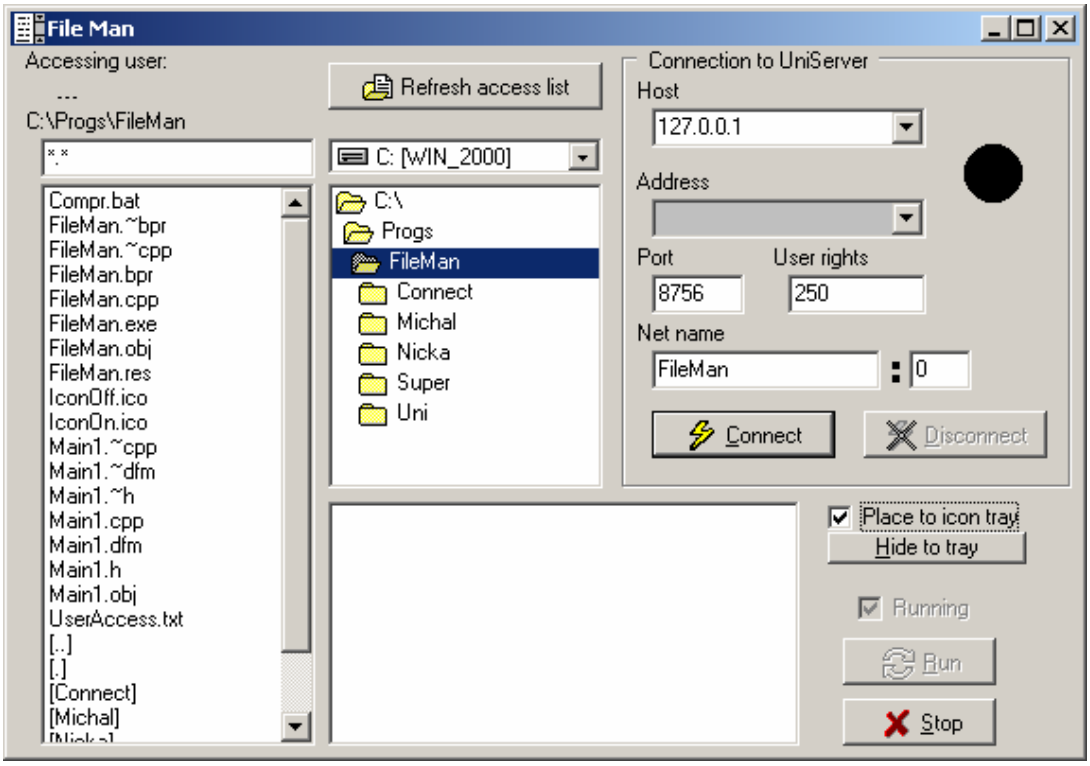
GETREMOTENETADDR

Popis: příkaz ke zjištění adresy pod kterou je UniBridge přihlášen do vzdálené sítě
Data příkazu: -
Odpověď: REMOTENETADDR
Data odpovědi: AnsiString Addr
Význam dat: Addr – adresa UniBridge v připojené síti

GETBRIDGEBRIDGELIST

Popis: příkaz ke zjištění seznamu adres klientu ve vzdálené síti, kteří jsou typu UniBridge
Data příkazu: -
Odpověď: BRIDGEBRIDGELIST
Data odpovědi: AnsiString AddrList
Význam dat: AddrList – seznam vzdálených UniBridge

9.3.10. File Man



Obrázek 9.3.11: Hlavní okno programu File Man

9.3.10.1. Podporované příkazy

DIR_R

Popis: zjištění obsahu aktuálního adresáře
Data příkazu: AnsiString Cmd
Význam dat: Cmd - příkaz
Odpověď: DIR_A
Data odpovědi: AnsiString Dir
AnsiString List
Význam dat: Dir – cesta aktuálního adresáře
List – seznam souborů a podadresářů

CD_R

Popis: změna adresáře
Data příkazu: AnsiString Dir
Význam dat: Dir – požadovaný adresář
Odpověď: CD_A
Data odpovědi: AnsiString Res
Význam dat: Res – výsledek požadavku na změnu adresáře

CD_R_HOME

Popis: změna adresáře na domovský adresář zasílajícího uživatele
Data příkazu: -

Význam dat: -
Odpověď: CD_A
Data odpovědi: AnsiString Res
Význam dat: Res – výsledek požadavku na změnu adresáře

GETFILEINFO

Popis: zjištění podrobných informací o souboru
Data příkazu: AnsiString FileName
Význam dat: FileName – jméno souboru
Odpověď: FILEINFO
Data odpovědi: AnsiString FileName
long Time
long Size
long Attr

GETFILE

Popis: download souboru
Data příkazu: AnsiString FileName
Význam dat: FileName – jméno souboru
Odpověď: PUTFILE
Data odpovědi: AnsiString FileName
long Size
Data
Význam dat: Data – obsah souboru uložený jako blok dat typu byte

DELFILE

Popis: smazání souboru
Data příkazu: AnsiString FileName
Význam dat: FileName – jméno souboru
Odpověď: FILEDELETED
Data odpovědi: AnsiString FileName

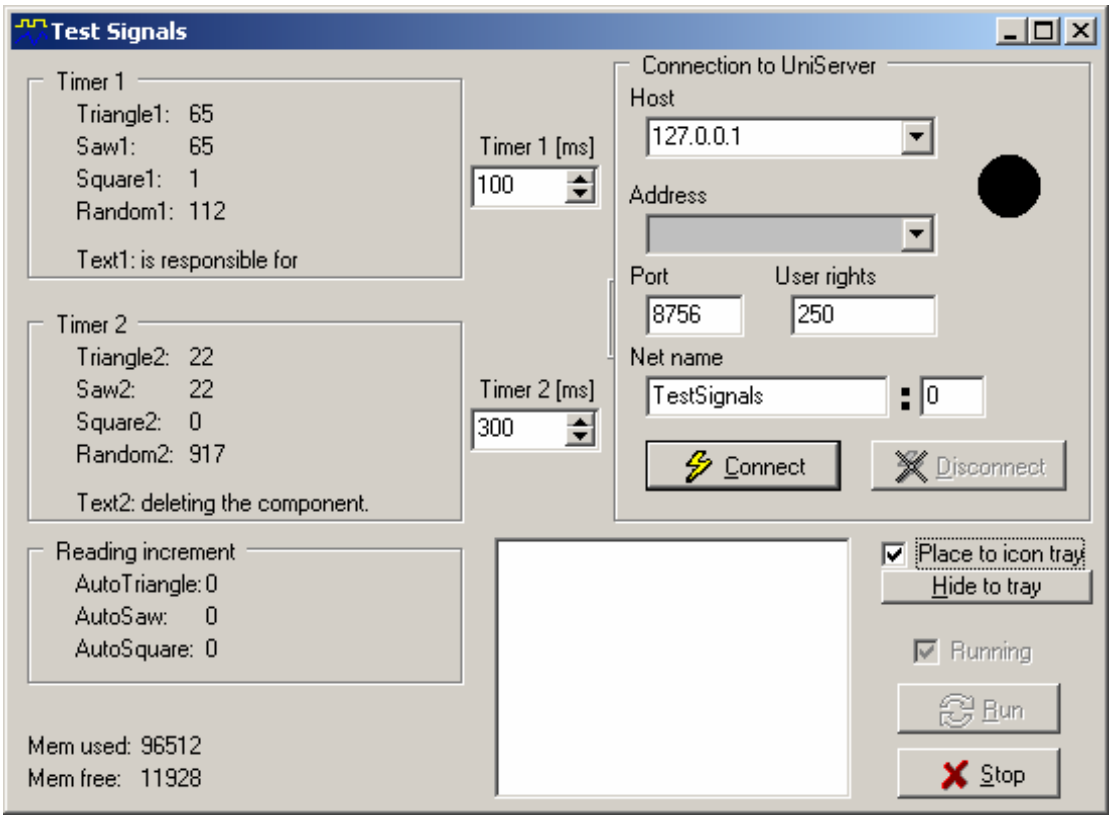
MKDIR

Popis: vytvoření adresáře
Data příkazu: AnsiString DirName
Význam dat: DirName – jméno adresáře
Odpověď: DIRCREATED
Data odpovědi: AnsiString DirName

RMDIR

Popis: vytvoření adresáře
Data příkazu: AnsiString DirName
Význam dat: DirName – jméno adresáře
Odpověď: DIRREMOVED
Data odpovědi: AnsiString DirName

9.3.11. Test Signals



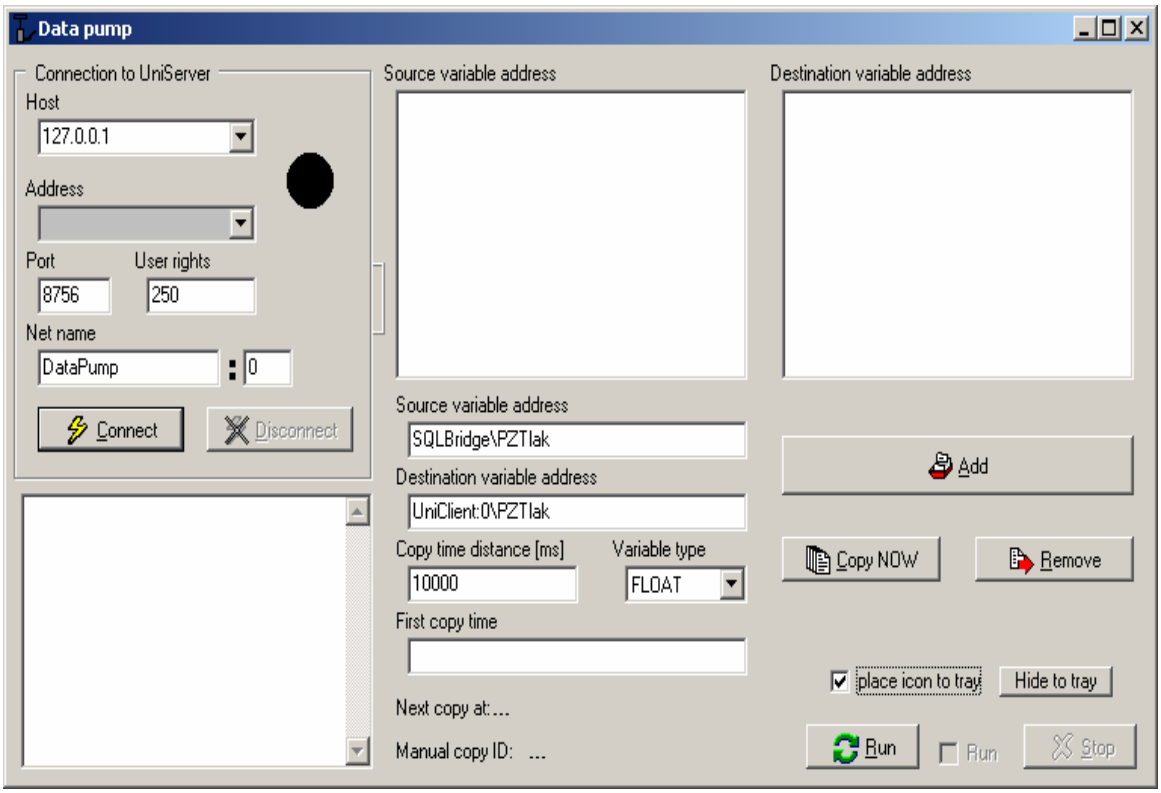
Obrázek 9.3.12: Hlavní okno programu Test Signals

9.3.11.1. Podporované příkazy

GETVAR

Příkaz je stejný jako u SQL Bridge

9.3.12. Data Pump

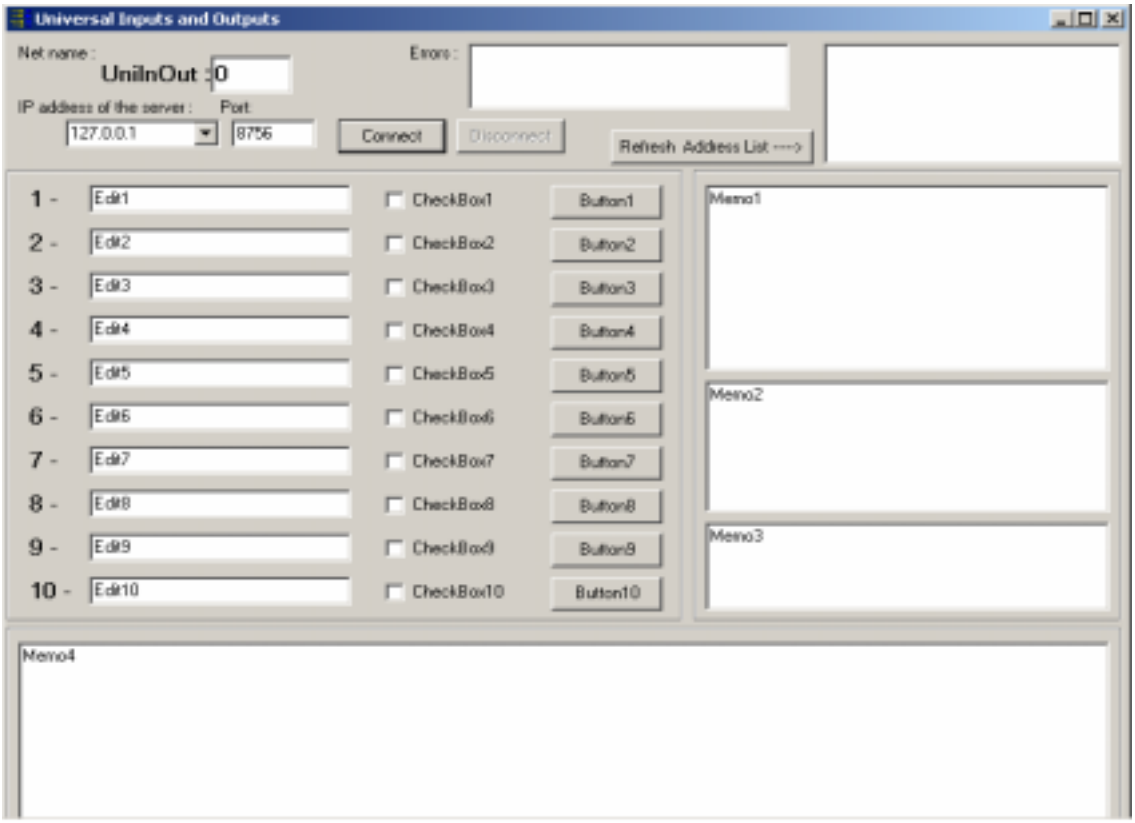


Obrázek 9.3.13: Hlavní okno programu Data Pump

9.3.12.1. Podporované příkazy

Tento klient nepodporuje žádné příkazy přijímané po síti UniNetvork od jiných klientů. Podporované jsou pouze systémové příkazy „GETCLIENTLIST“ a „ERRORMESSAGE“.

9.3.13. Klient InOut



Obrázek 9.3.14: Hlavní okno programu InOut

9.3.13.1. Podporované příkazy

GETVAR

Příkaz je stejný jako u SQL Bridge

PUTVAR

Příkaz je stejný jako u SQL Bridge

BUTTONPRESSED

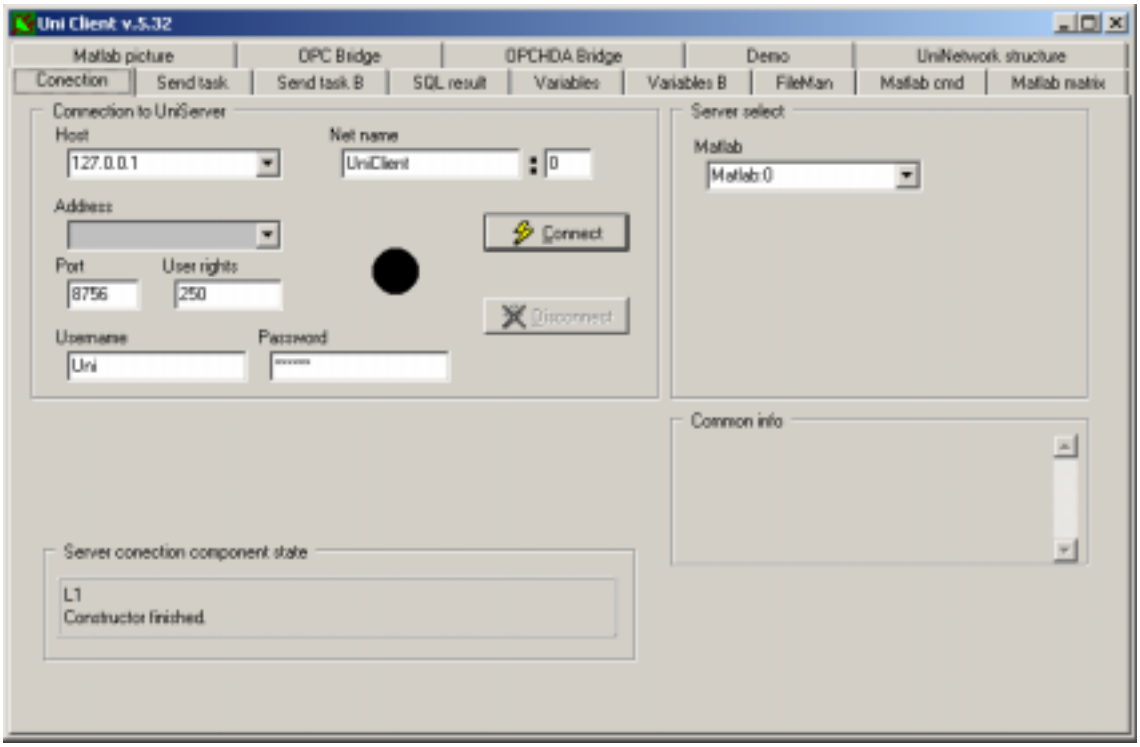
Popis: příkaz je odesílán klientem v případě, že je zmáčknuto tlačítko Button1 až 10.

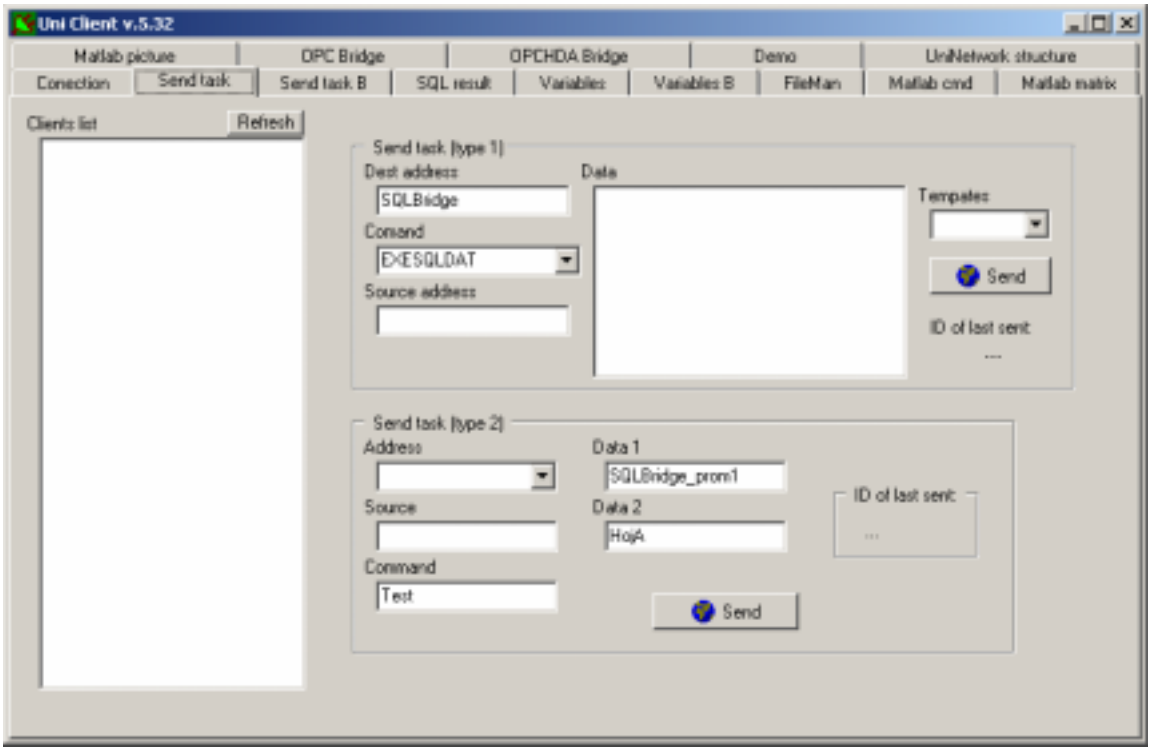
Data příkazu: -

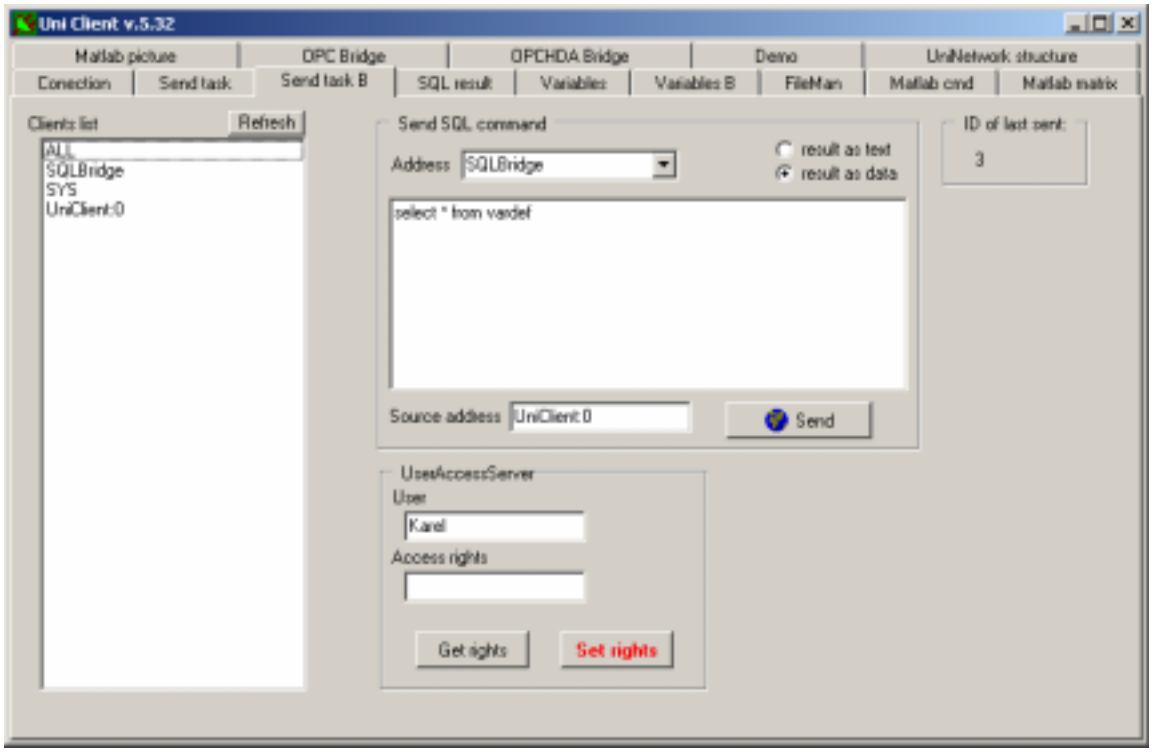
Význam dat: -

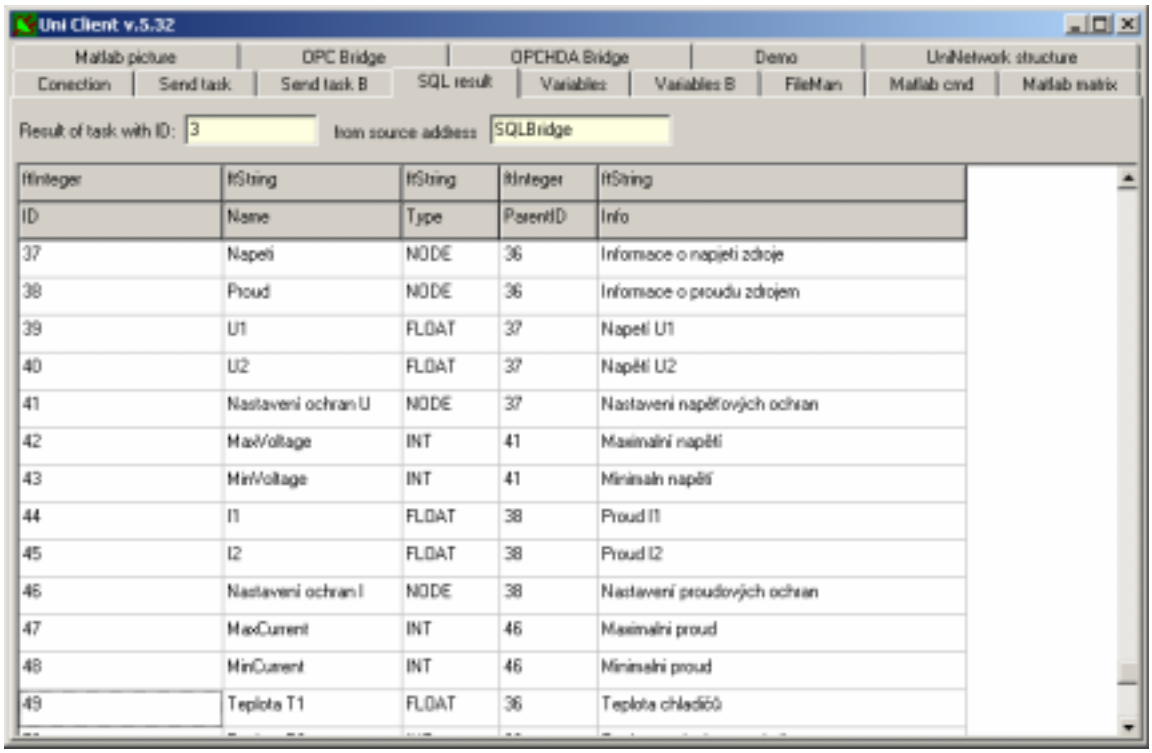
Odpověď: -

9.3.14. UniClient

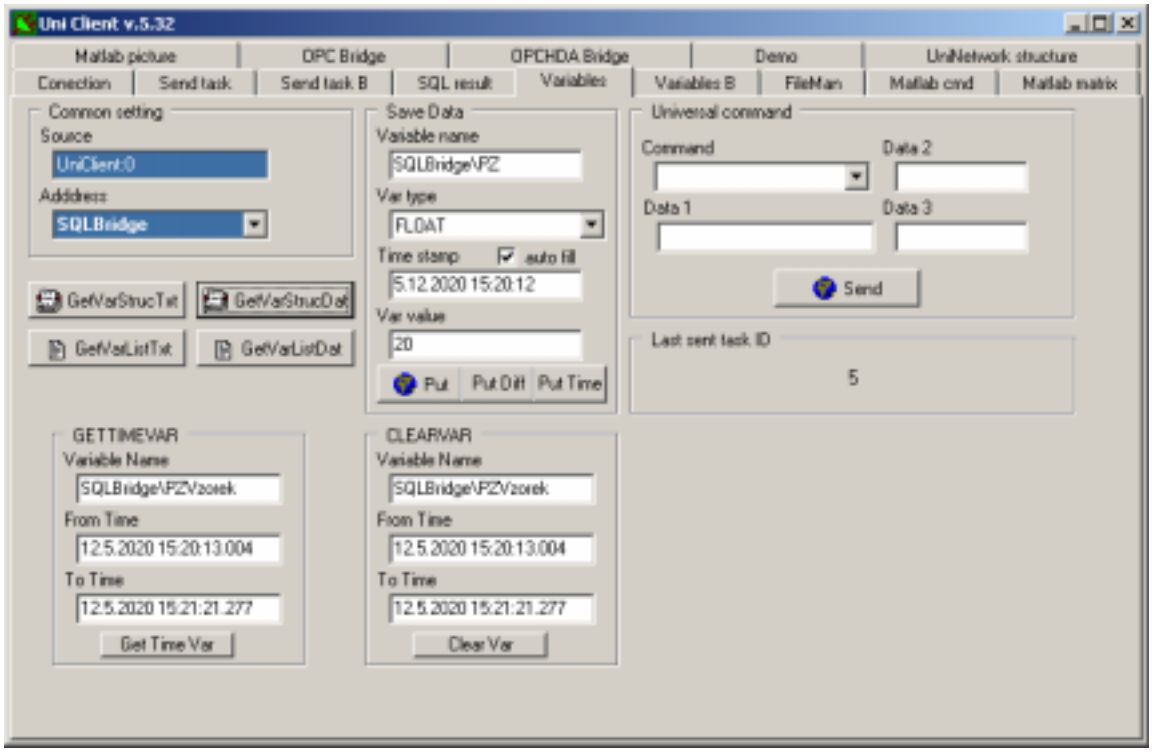


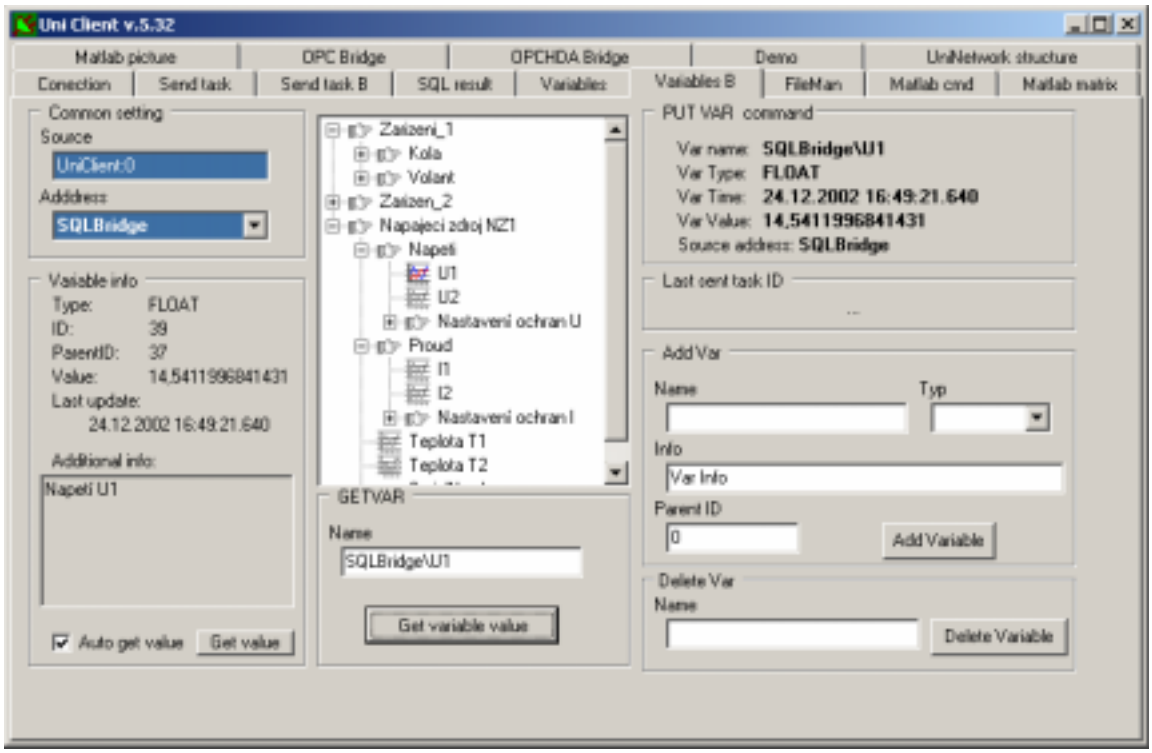


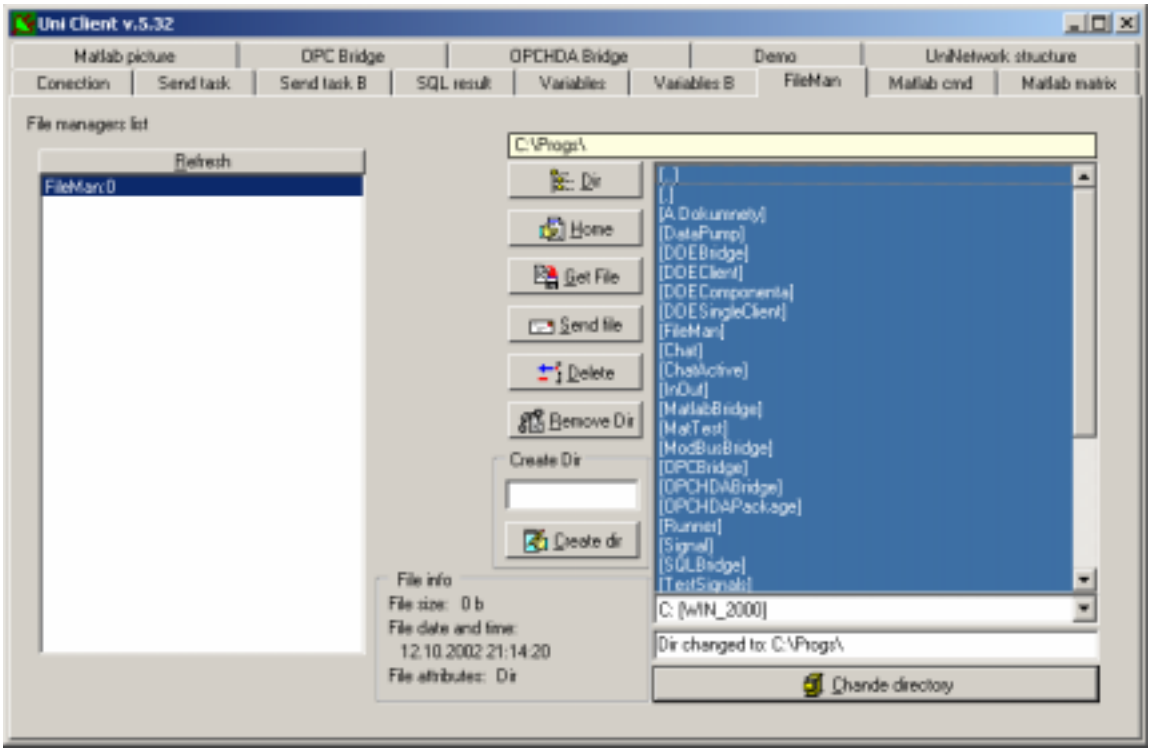


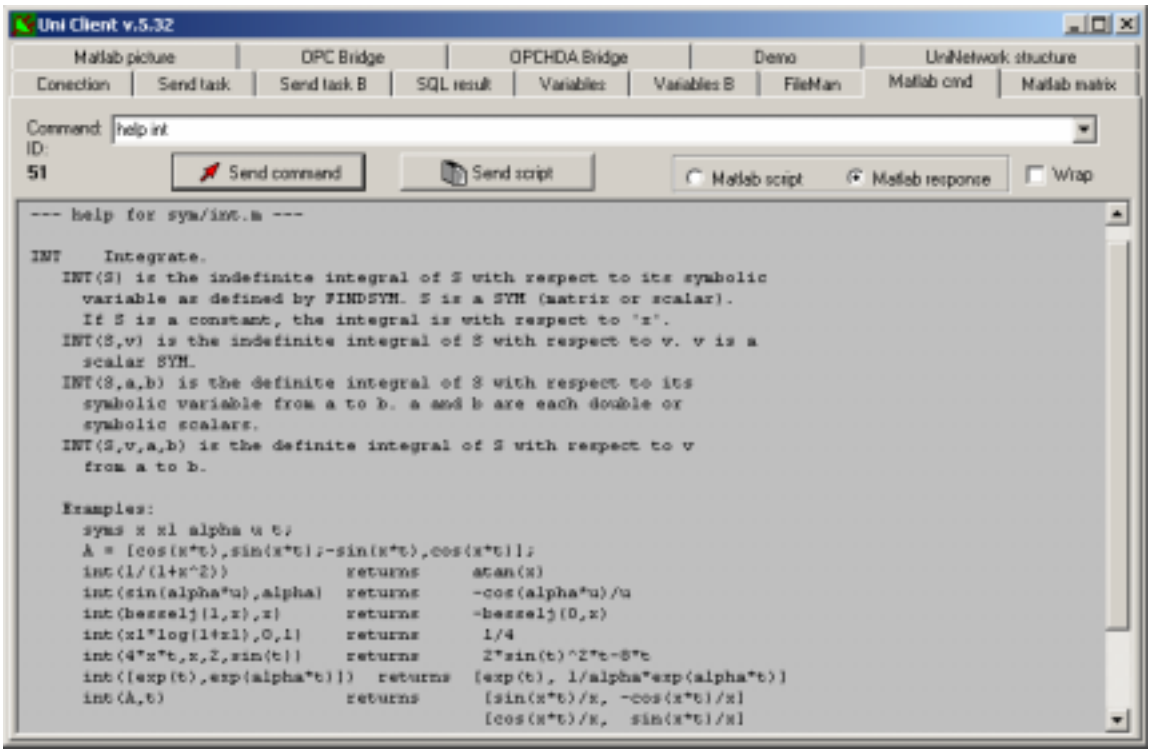


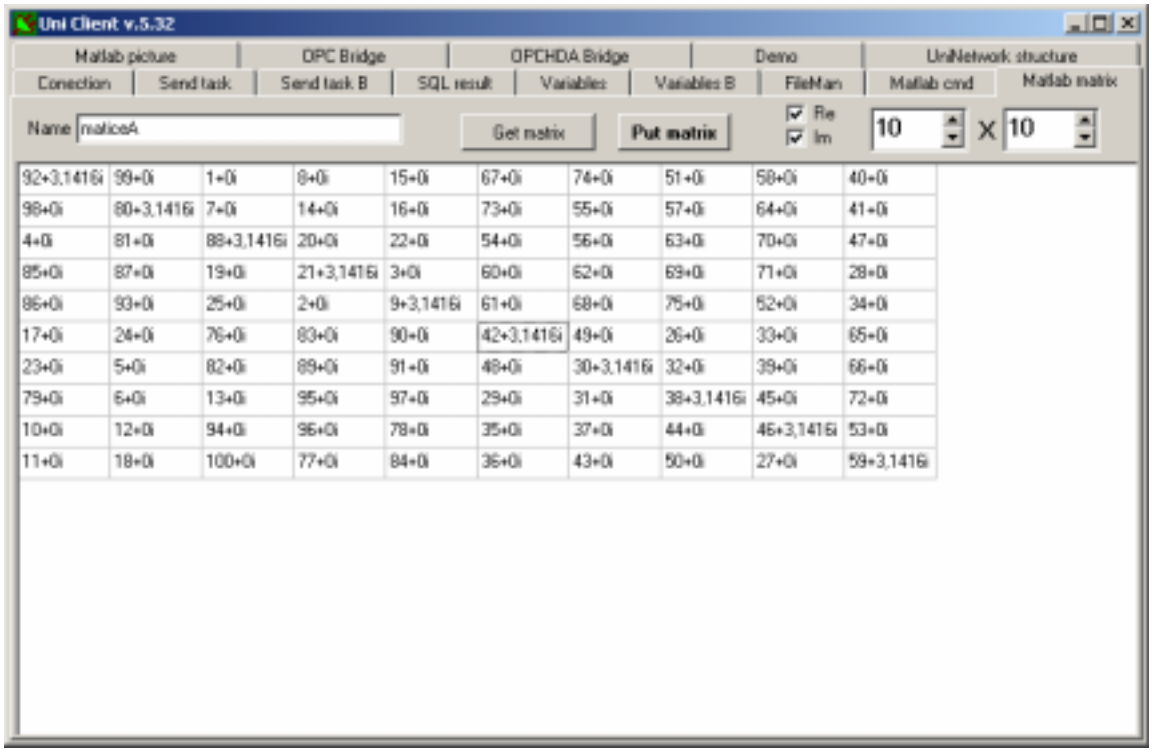
ftInteger	ftString	ftString	ftInteger	ftString
ID	Name	Type	ParentID	Info
37	Napeti	NODE	36	Informace o napjetí zdroje
38	Proud	NODE	36	Informace o proudu zdrojem
39	U1	FLOAT	37	Napeti U1
40	U2	FLOAT	37	Napětí U2
41	Nastaveni ochran U	NODE	37	Nastavení napěťových ochran
42	MaxVoltage	INT	41	Maximální napětí
43	MinVoltage	INT	41	Minimaln napětí
44	I1	FLOAT	38	Proud I1
45	I2	FLOAT	38	Proud I2
46	Nastaveni ochran I	NODE	38	Nastavení proudových ochran
47	MaxCurrent	INT	46	Maximální proud
48	MinCurrent	INT	46	Minimální proud
49	Teplota T1	FLOAT	36	Teplota chladičů

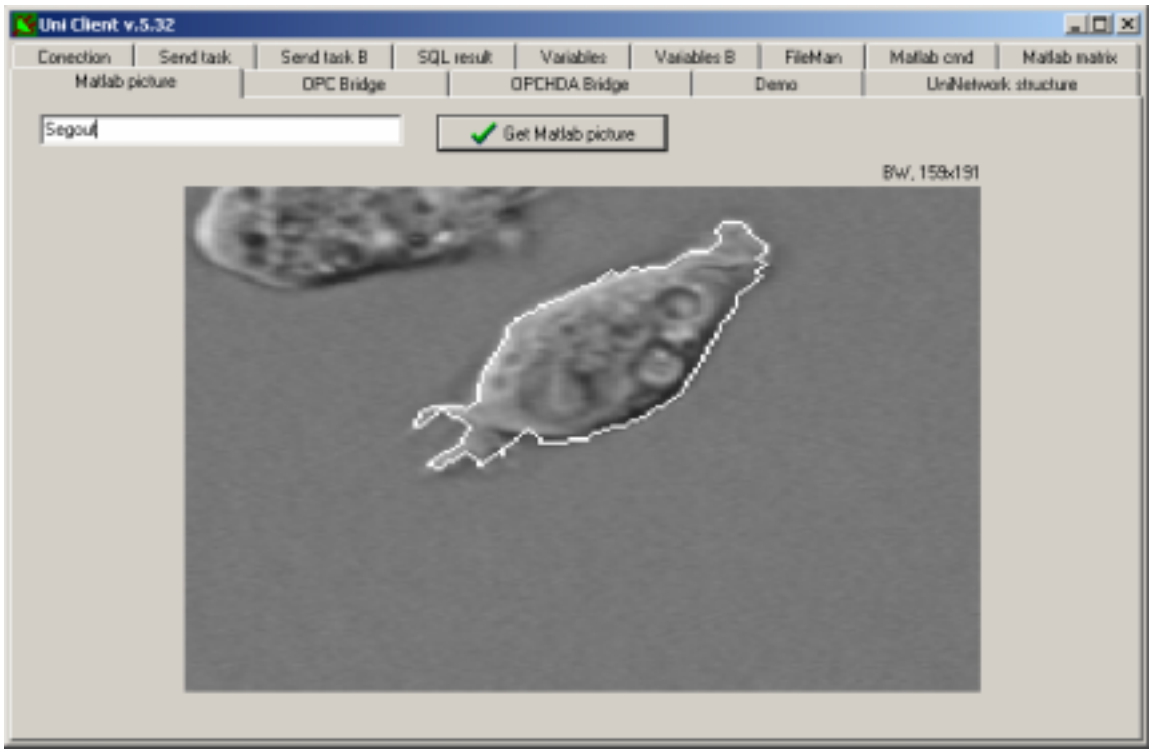

Uni Client v.5.32
Matlab picture
OPC Bridge
OPCHDA Bridge
Demo
UniNetwork structure
Conection
Send task
Send task B
SQL result
Variables
Variables B
FileMan
Matlab cmd
Matlab matrix
Common setting
Source
UniClient:0
Address
SQLBridge
GetVarStrucTxt
GetVarStrucDat
GetVarListTxt
GetVarListDat
Save Data
Variable name
SQLBridge\PZ
Var type
FLOAT
Time stamp
auto fill
5.12.2020 15:20:12
Var value
20
Put
Put Diff
Put Time
Universal command
Command
Data 2
Data 1
Data 3
Send
Last sent task ID
5
GETTIMEVAR
Variable Name
SQLBridge\PZVzorek
From Time
12.5.2020 15:20:13.004
To Time
12.5.2020 15:21:21.277
Get Time Var
CLEARVAR
Variable Name
SQLBridge\PZVzorek
From Time
12.5.2020 15:20:13.004
To Time
12.5.2020 15:21:21.277
Clear Var

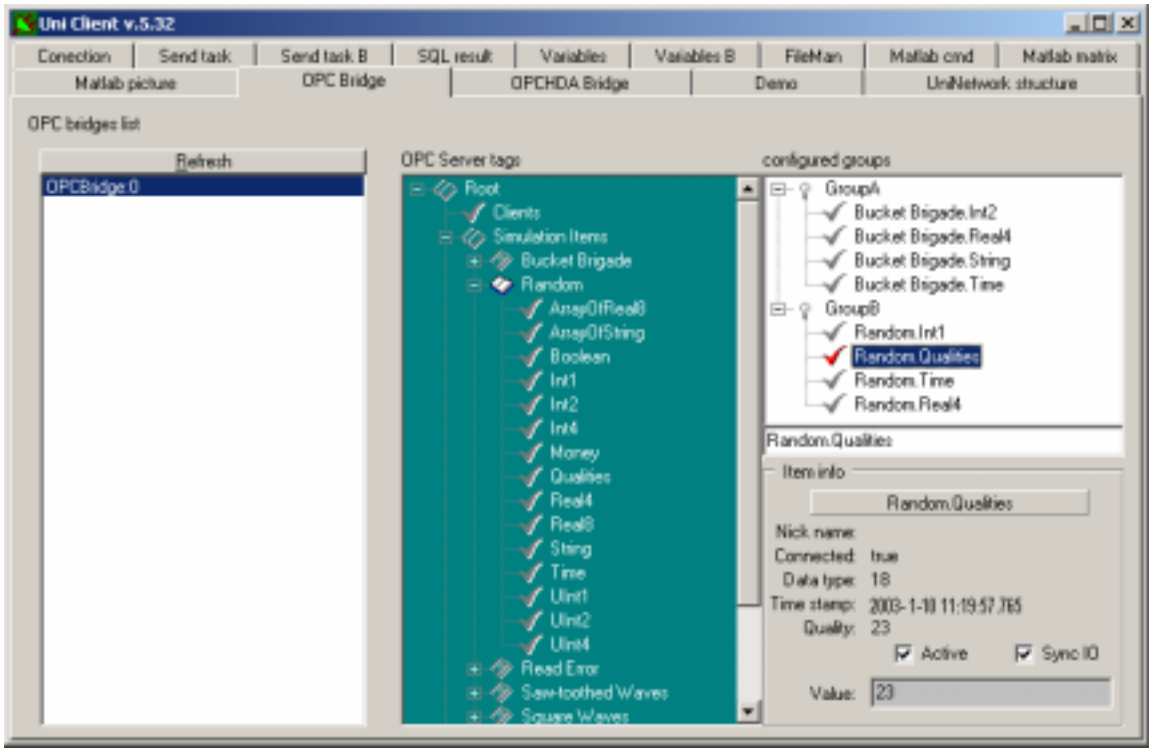

Uni Client v.5.32
Matlab picture
OPC Bridge
OPCHDA Bridge
Demo
UniNetwork structure
Conection
Send task
Send task B
SQL result
Variables
Variables B
FileMan
Matlab cmd
Matlab matrix
Common setting
Source
UniClient:0
Address
SQLBridge
Variable info
Type: FLOAT
ID: 39
ParentID: 37
Value: 14,5411996841431
Last update:
24.12.2002 16:49:21.640
Additional info:
Napeti U1
Auto get value
Get value
Zaizeni_1
Kola
Volant
Zaizen_2
Napajeci zdroj NZ1
Napeti
U1
U2
Nastaveni ochran U
Proud
I1
I2
Nastaveni ochran I
Teplota T1
Teplota T2
GETVAR
Name
SQLBridge\U1
Get variable value
PUT VAR command
Var name: SQLBridge\U1
Var Type: FLOAT
Var Time: 24.12.2002 16:49:21.640
Var Value: 14,5411996841431
Source address: SQLBridge
Last sent task ID
Add Var
Name
Typ
Info
Var Info
Parent ID
0
Add Variable
Delete Var
Name
Delete Variable

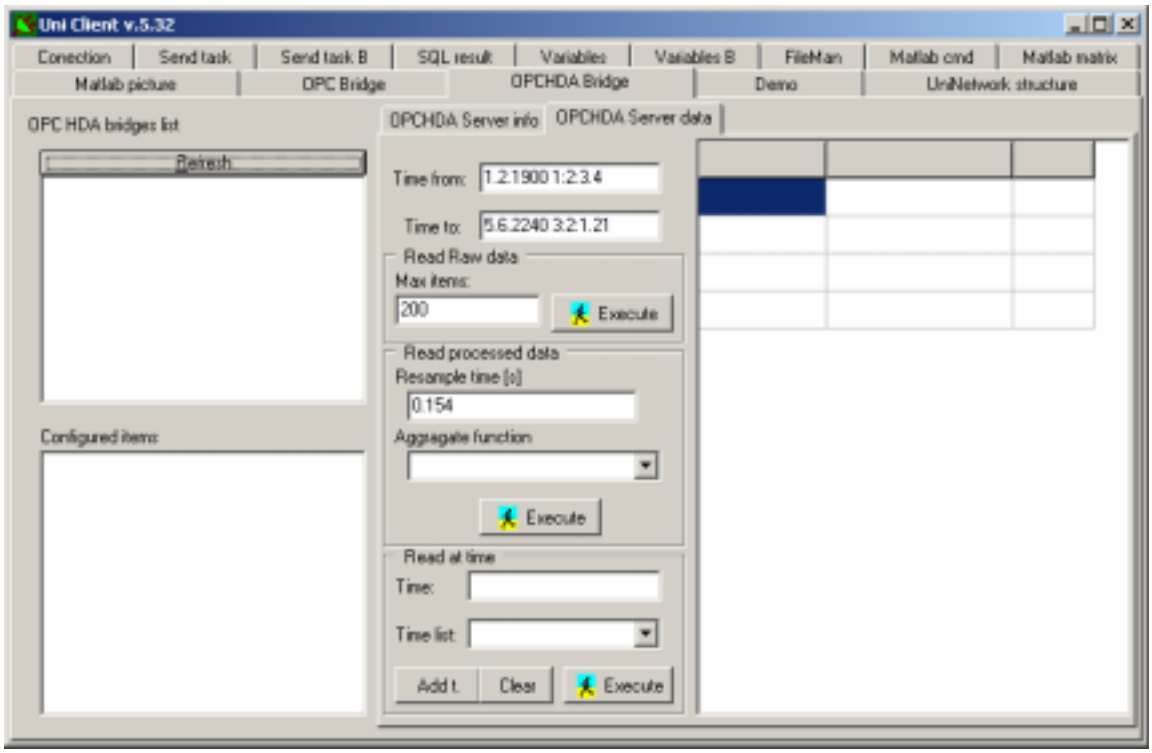

Uni Client v.5.32
Matlab picture
OPC Bridge
OPCHDA Bridge
Demo
UniNetwork structure
Conection
Send task
Send task B
SQL result
Variables
Variables B
FileMan
Matlab cmd
Matlab matrix
File manager list
Refresh
FileMan:0
C:\Progs\
Dir
Home
Get File
Send file
Delete
Remove Dir
Create Dir
Create dir
File info
File size: 0 b
File date and time:
12.10.2002 21:14:20
File attributes: Dir
C: [WIN_2000]
Dir changed to: C:\Progs\
Chande directory

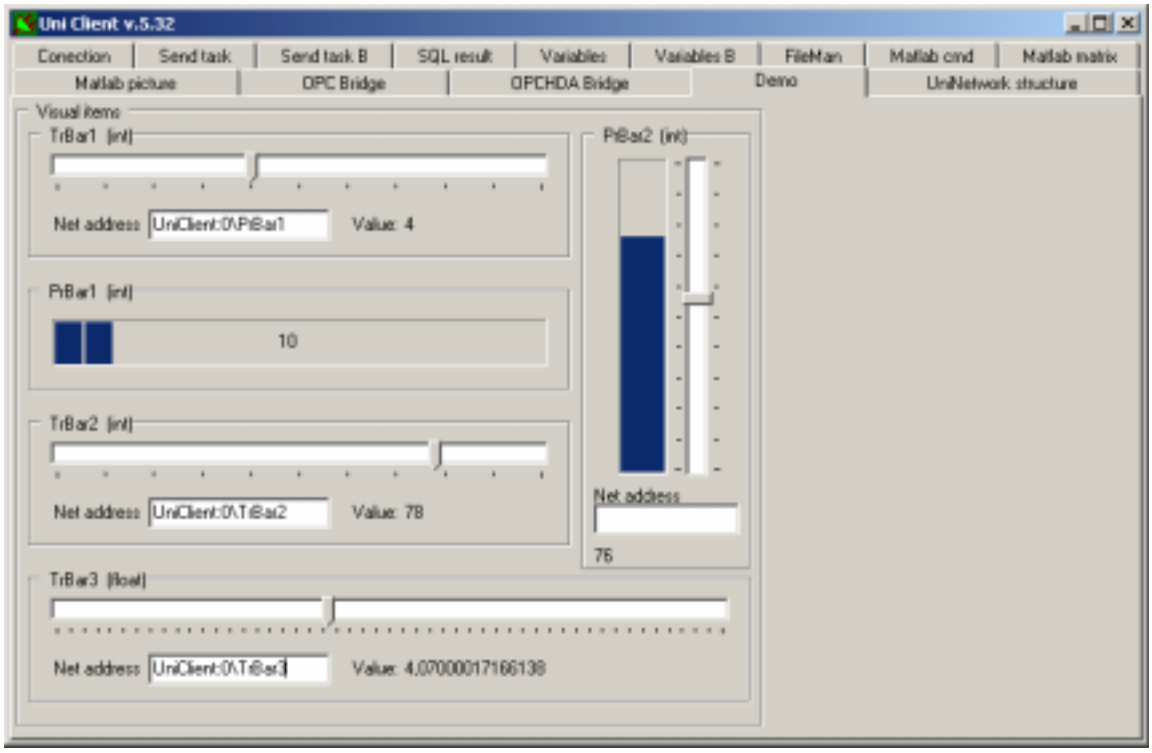

Uni Client v.5.32
Matlab picture
OPC Bridge
OPCHDA Bridge
Demo
UniNetwork structure
Conection
Send task
Send task B
SQL result
Variables
Variables B
FileMan
Matlab cmd
Matlab matrix
Command: help int
ID:
51
Send command
Send script
Matlab script
Matlab response
Wrap
--- help for sym/int.m ---
INT Integrate.
INT(S) is the indefinite integral of S with respect to its symbolic
variable as defined by FINDSYM. S is a SYM (matrix or scalar).
If S is a constant, the integral is with respect to 'x'.
INT(S,v) is the indefinite integral of S with respect to v. v is a
scalar SYM.
INT(S,a,b) is the definite integral of S with respect to its
symbolic variable from a to b. a and b are each double or
symbolic scalars.
INT(S,v,a,b) is the definite integral of S with respect to v
from a to b.
Examples:
syms x xl alpha u t;
A = [cos(x*t),sin(x*t);-sin(x*t),cos(x*t)];
int(1/(1+x^2)) returns atan(x)
int(sin(alpha*u),alpha) returns -cos(alpha*u)/u
int(besselj(1,x),x) returns -besselj(0,x)
int(xl*log(1+xl),0,1) returns 1/4
int(4*x*t,x,2,sin(t)) returns 2*sin(t)^2*t-8*t
int([exp(t),exp(alpha*t)]) returns [exp(t), 1/alpha*exp(alpha*t)]
int(A,t) returns [sin(x*t)/x, -cos(x*t)/x]
[cos(x*t)/x, sin(x*t)/x]

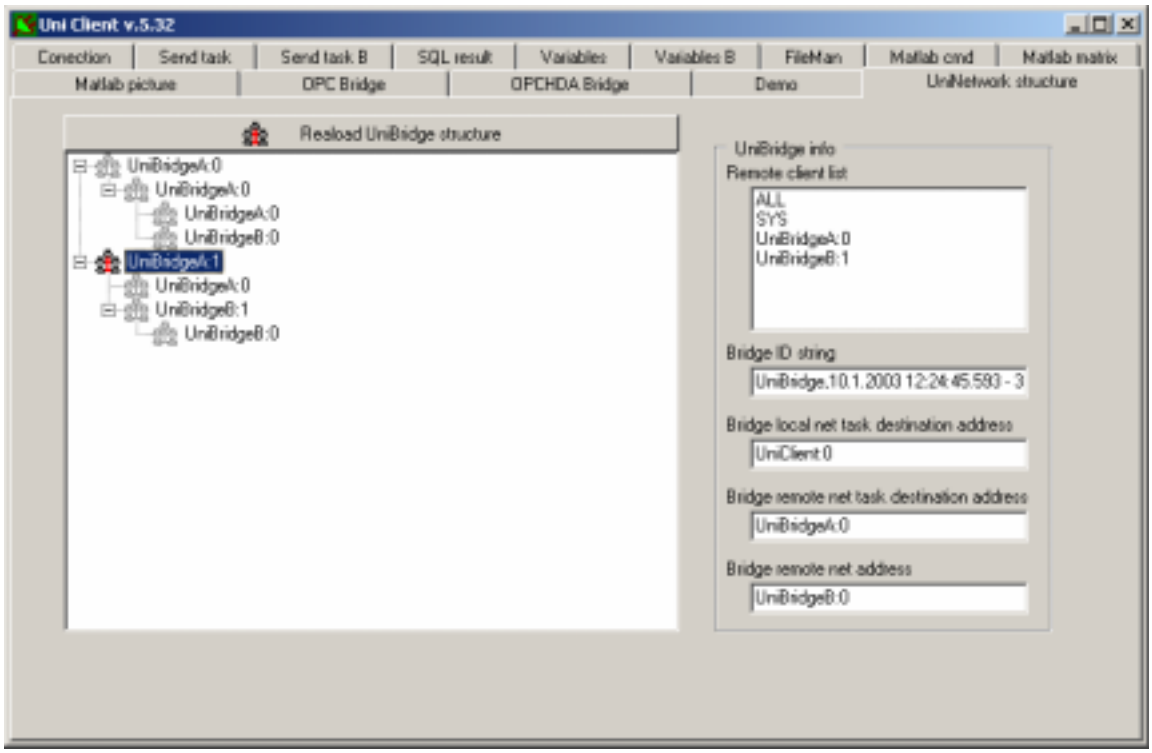

Uni Client v.5.32
Matlab picture
OPC Bridge
OPCHDA Bridge
Demo
UniNetwork structure
Conection
Send task
Send task B
SQL result
Variables
Variables B
FileMan
Matlab cmd
Matlab matrix
Name maticeA
Get matrix
Put matrix
Re
Im
10
X
10
92+3,1416i 99+0i 1+0i 8+0i 15+0i 67+0i 74+0i 51+0i 58+0i 40+0i
98+0i 80+3,1416i 7+0i 14+0i 16+0i 73+0i 55+0i 57+0i 64+0i 41+0i
4+0i 81+0i 88+3,1416i 20+0i 22+0i 54+0i 56+0i 63+0i 70+0i 47+0i
85+0i 87+0i 19+0i 21+3,1416i 3+0i 60+0i 62+0i 69+0i 71+0i 28+0i
86+0i 93+0i 25+0i 2+0i 9+3,1416i 61+0i 68+0i 75+0i 52+0i 34+0i
17+0i 24+0i 76+0i 83+0i 90+0i 42+3,1416i 49+0i 26+0i 33+0i 65+0i
23+0i 5+0i 82+0i 89+0i 91+0i 48+0i 30+3,1416i 32+0i 39+0i 66+0i
79+0i 6+0i 13+0i 95+0i 97+0i 29+0i 31+0i 38+3,1416i 45+0i 72+0i
10+0i 12+0i 94+0i 96+0i 78+0i 35+0i 37+0i 44+0i 46+3,1416i 53+0i
11+0i 18+0i 100+0i 77+0i 84+0i 36+0i 43+0i 50+0i 27+0i 59+3,1416i


Uni Client v.5.32
Conection
Send task
Send task B
SQL result
Variables
Variables B
FileMan
Matlab cmd
Matlab matrix
Matlab picture
OPC Bridge
OPCHDA Bridge
Demo
UniNetwork structure
Segout
Get Matlab picture
BW, 158x191


Uni Client v.5.32
Conection
Send task
Send task B
SQL result
Variables
Variables B
FileMan
Matlab cmd
Matlab matrix
Matlab picture
OPC Bridge
OPCHDA Bridge
Demo
UniNetwork structure
OPC bridges list
Refresh
OPCBridge:0
OPC Server tags
Root
Clients
Simulation Items
Bucket Brigade
Random
ArrayOfReal8
ArrayOfString
Boolean
Int1
Int2
Int4
Money
Qualities
Real4
Real8
String
Time
UInt1
UInt2
UInt4
Read Error
Saw-toothed Waves
configured groups
GroupA
Bucket Brigade.Int2
Bucket Brigade.Real4
Bucket Brigade.String
Bucket Brigade.Time
GroupB
Random.Int1
Random.Qualities
Random.Time
Random.Real4
Random.Qualities
Item info
Random.Qualities
Nick name:
Connected: true
Data type: 18
Time stamp: 2003-1-10 11:19:57.765
Quality: 23
Active
Sync IO
Value: 23

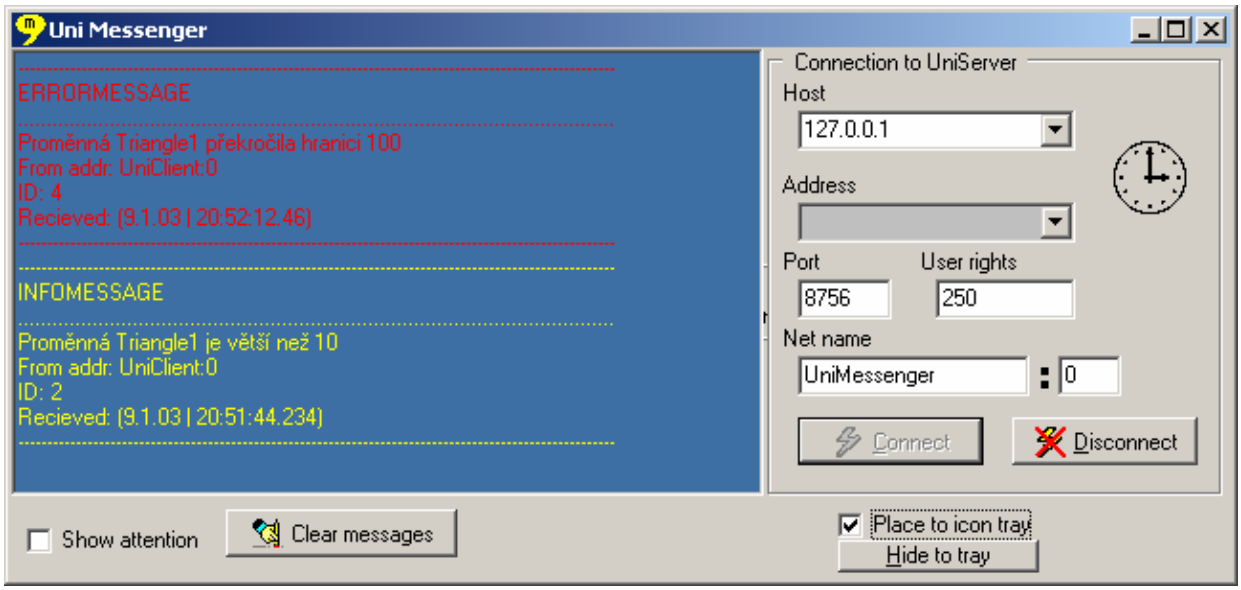

Uni Client v.5.32
Conection
Send task
Send task B
SQL result
Variables
Variables B
FileMan
Matlab cmd
Matlab matrix
Matlab picture
OPC Bridge
OPCHDA Bridge
Demo
UniNetwork structure
OPC HDA bridges list
Refresh
Configured items:
OPCHDA Server info
OPCHDA Server data
Time from: 1.2.1900 1:2:3.4
Time to: 5.6.2240 3:2:1.21
Read Raw data
Max items:
200
Execute
Read processed data
Resample time [s]
0.154
Aggregate function
Execute
Read at time
Time:
Time list:
Add t.
Clear
Execute





Obrázek 9.3.15: Okna aplikace UniClient

9.3.15. UniMessenger



Obrázek 9.3.16: Hlavní okno programu UniMessenger

9.3.15.1. Podporované příkazy

ERRORMESSAGE

Popis: zobrazení chybové hlášky v okně (červeně)
Data příkazu: AnsiString Text
Význam dat: Text – text hlášení
Odpověď: -

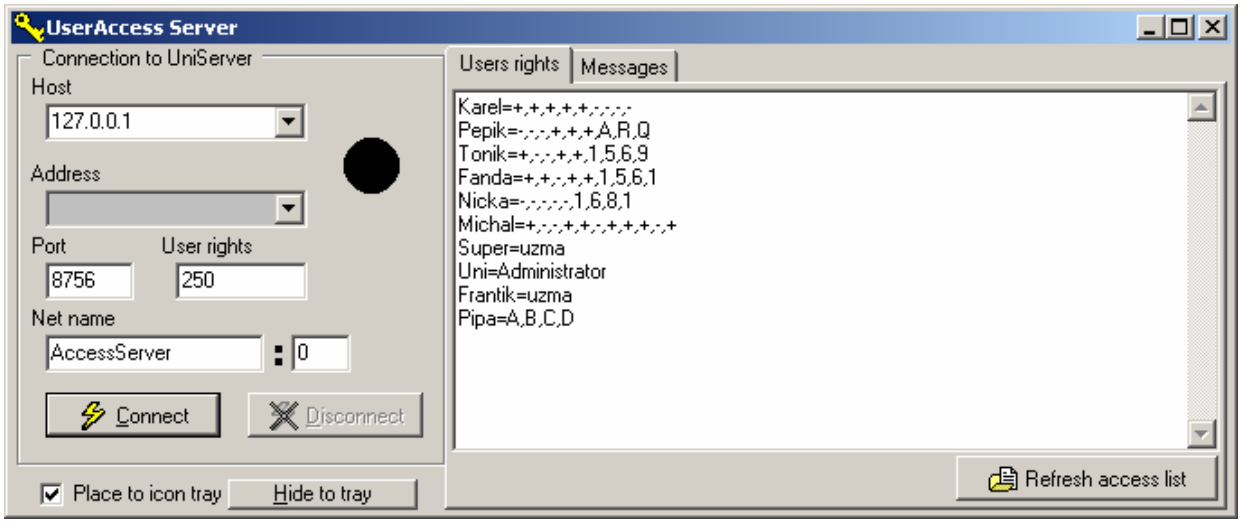
INFOMESSAGE

Popis: zobrazení informace v okně (žlutě)
Data příkazu: AnsiString Text
Význam dat: Text – text informace
Odpověď: -

YESNOMESSAGE

Popis: zobrazení dialogu s o otázkou a dvěmi „YES“ a „NO“
Data příkazu: AnsiString Text
Význam dat: Text – text otázky
Odpověď: YESNOANSWER
Data odpovědi: bool Res
Význam dat: Res – odpověď na otázku (true/false)

9.3.16. Klient User Access Serer



Obrázek 9.3.17: Hlavní okno programu UserAccessServer

9.3.16.1. Podporované příkazy

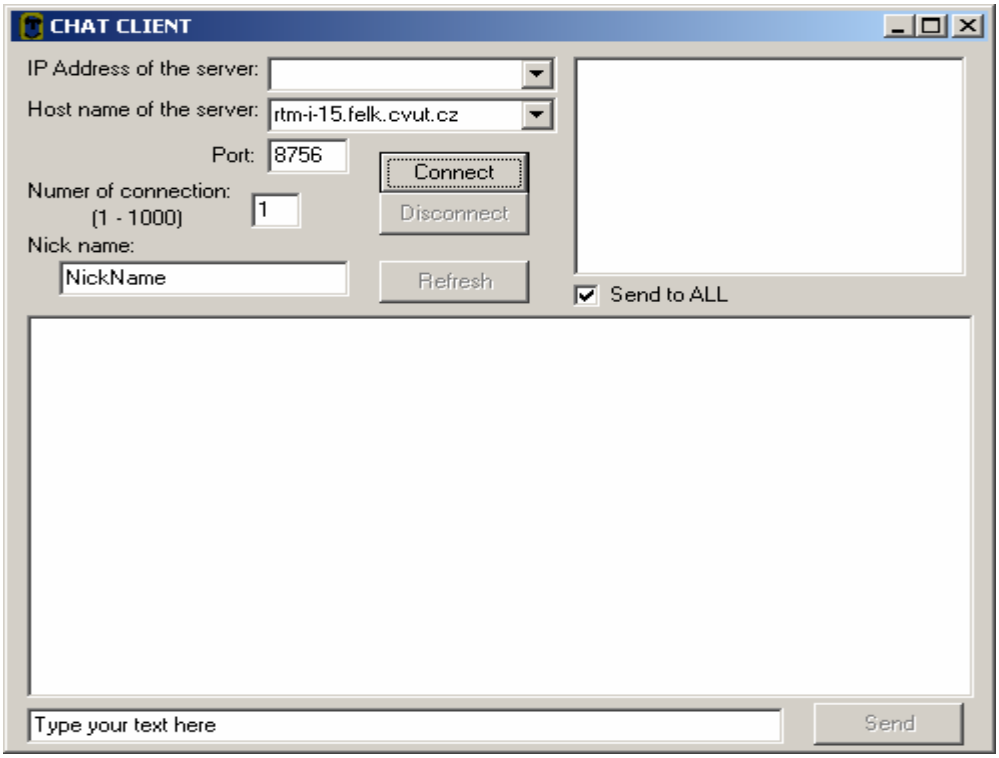
GETUSERRIGHTS

Popis: příkaz pro získání proměnné s definicí práv specifikovaného uživatele
Data příkazu: AnsiString UserName
Význam dat: UserName – jméno uživatele
Odpověď: USERRIGHTS
Data odpovědi: AnsiString UserRights
Význam dat: UserRights – textový řetězec s právy uživatele

SETUSERRIGHTS

Popis: příkaz pro nastavení proměnné s definicí práv specifikovaného uživatele
Data příkazu: AnsiString UserName
AnsiString UserRights
Význam dat: UserName – jméno uživatele
UserRights – textový řetězec s právy uživatele
Odpověď: SETRIGHTS_OK
Data odpovědi: AnsiString UserRights
Význam dat: UserRights – textový řetězec s právy uživatele

9.3.17. Klinet UniChat



Obrázek 9.3.18: Hlavní okno programu UniChat

9.3.17.1. Podporované příkazy

GETNICKNAME

Popis: příkaz pro získání přezdívky uživatele UniChat
Data příkazu: -
Význam dat: -
Odpověď: MYNICKNAME
Data odpovědi: AnsiString NickName
Význam dat: NickName – přezdívka uživatele

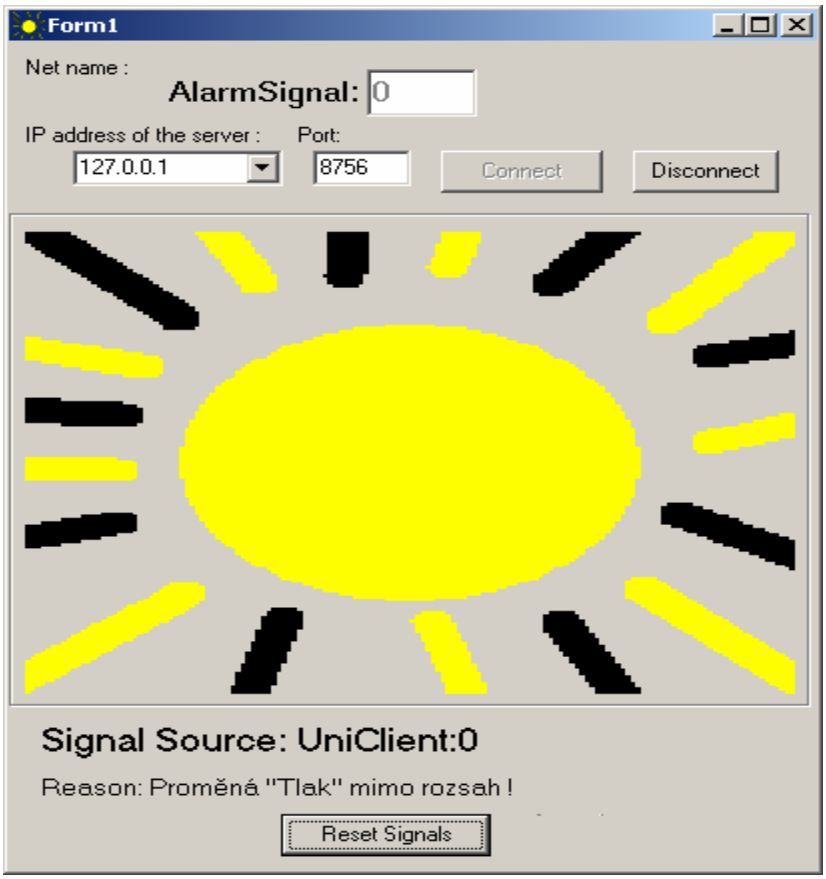
CLIENTLOGOUT

Popis: příkaz informující UniChat, že některý klient se odhlásil
Data příkazu: -
Význam dat: -
Odpověď: -

CHATSTRING

Popis: příkaz pro zobrazení textu konverzace
Data příkazu: AnsiString Text
Význam dat: Text – text konverzace
Odpověď: -

9.3.18. Klient Signal



Obrázek 9.3.19: Hlavní okno programu Signal

9.3.18.1. Podporované příkazy

SETALARM

Popis: příkaz pro zobrazení akustického a vizuálního alarmu
Data příkazu: AnsiString Reas
Význam dat: Reas – důvod pro spuštění alarmu, zobrazí se v okně
Odpověď: SETALARM
Data odpovědi: bool Resp
Význam dat: Resp – vždy „true“

RESETALARM

Popis: příkaz pro zrušení akustického i vizuálního alarmu
Data příkazu: AnsiString Reas
Význam dat: Reas – důvod pro zrušení alarmu, zobrazí se v okně
Odpověď: RESETALARM
Data odpovědi: bool Resp
Význam dat: Resp – vždy „true“

SETVISUALALARM

Popis: příkaz pro zobrazení vizuálního alarmu
Data příkazu: AnsiString Reas
Význam dat: Reas – důvod pro spuštění alarmu, zobrazí se v okně
Odpověď: SETVISUALALARM
Data odpovědi: bool Resp
Význam dat: Resp – vždy „true“

RESETVISUALALARM

Popis: příkaz pro zrušení vizuálního alarmu
Data příkazu: AnsiString Reas
Význam dat: Reas – důvod pro zrušení alarmu, zobrazí se v okně
Odpověď: RESETVISUALALARM
Data odpovědi: bool Resp
Význam dat: Resp – vždy „true“

SETACUSTICALARM

Popis: příkaz pro spuštění akustického alarmu
Data příkazu: AnsiString Reas
Význam dat: Reas – důvod pro spuštění alarmu, zobrazí se v okně
Odpověď: SETACUSTICALARM
Data odpovědi: bool Resp
Význam dat: Resp – vždy „true“

RESETACUSTICALARM

Popis: příkaz pro zrušení akustického alarmu
Data příkazu: AnsiString Reas
Význam dat: Reas – důvod pro zrušení alarmu, zobrazí se v okně
Odpověď: RESETACUSTICALARM
Data odpovědi: bool Resp
Význam dat: Resp – vždy „true“

9.3.19. Klient Runner

9.3.19.1. Ukázka řídicího kódu

```
// Zaslání příkazu pro klienta typu Matlab pro provedení příkazu „picx ´= ...“.
Aplikace Matlab je zde používana pomocí klienta Matlab Bridge.
  UniConnection1->DesAddress = "Matlab:2";
  UniConnection1->SrcAddress = "Runner";
  UniConnection1->Command = "MATLABDOCOMMAND";
  UniConnection1->WriteDatString("picx = avfm(1)*255;");
  UniConnection1->SendTask();

  UniConnection1->DesAddress = "Matlab:2";
  UniConnection1->SrcAddress = "Runner";
  UniConnection1->Command = "MATLABDOCOMMAND";
  UniConnection1->WriteDatString("pic1 = fix(picx);");
```

```
  UniConnection1->SendTask();


  UniConnection1->DesAddress = "Matlab:2";
  UniConnection1->SrcAddress = "Runner";
  UniConnection1->Command = "MATLABDOCOMMAND";
  UniConnection1->WriteDatString("pic2 = fix(rgb2gray(picx));");
  UniConnection1->SendTask();

// Zaslání požadavku o obrázek uložený v Matlabu, odpověď z Matlabu je zaslána na
klientovi jménem UniClient:10, který obrázek zobrazí.
  UniConnection1->DesAddress = "Matlab:2";
  UniConnection1->SrcAddress = "UniClient:10";
  UniConnection1->Command = "MATLABGETPICTURE";
  UniConnection1->WriteDatString("pic1");
  UniConnection1->SendTask();

  UniConnection1->DesAddress = "Matlab:2";
  UniConnection1->SrcAddress = "UniClient:11";
  UniConnection1->Command = "MATLABGETPICTURE";
  UniConnection1->WriteDatString("pic2");
  UniConnection1->SendTask();

// Zaslání požadavku na uložení hodnoty proměnné do Matlabu, který je připojen pomocí
ActiveX objektu (v Matlabu je spuštěn řídicí skript).
  UniConnection1->DesAddress = "MatAkt:1";
  UniConnection1->SrcAddress = "Runner";
  UniConnection1->Command = "PUTDATA";
  UniConnection1->WriteDatString("Var1");
  UniConnection1->AddDatString("INT");
  UniConnection1->AddDatString(Time().TimeString());
  UniConnection1->AddDatLong(D1[ic]);
  UniConnection1->SendTask();

// Spuštění skriptu v aplikaci Matlab
  UniConnection1->DesAddress = "Matlab:1";
  UniConnection1->SrcAddress = "Runner";
  UniConnection1->Command = "MATLABDOCOMMAND";
  UniConnection1->WriteDatString("process");
  UniConnection1->SendTask();

// Uložení hodnoty proměnné do sešitu v MS Excel, kde je použit objekt ActiveX
  UniConnection1->DesAddress = "Exl:1";
  UniConnection1->SrcAddress = "Runner";
  UniConnection1->Command = "PUTDATA";
  UniConnection1->WriteDatString("Var1");
  UniConnection1->AddDatString("INT");
  UniConnection1->AddDatString(Time().TimeString());
  UniConnection1->AddDatLong(D1[ic]*4+1);
  UniConnection1->SendTask();

// Zobrazení aktuálního času v okně klienta UniInOut:0
  UniConnection1->DesAddress = "UniInOut:0";
  UniConnection1->SrcAddress = "Runner";
  UniConnection1->Command = "SETEDIT";
  UniConnection1->WriteDatLong(1);
  UniConnection1->AddDatString(Time().TimeString());
  UniConnection1->SendTask();
```

9.4. CD ROM

Obsah přiloženého CD ROM:

/exe - programy balíku aplikací UniNetwork

/src - zdrojové kódy k aplikacím

/doc - text diplomové práce ve formátech MS Word XP a PDF, dokumentace k UniNetwork

/examples - soubory příkladů popsané výše v textu

/database - ukázková databáze pro program SQL Bridge, formátu MySQL, verze: 3.23.38

/mysql - instalační program MySQL verze: 3.23.38

/zaloha1 - obsahuje zálohu všech souborů z kořenového adresáře pro případ poškození CD.

readme.txt - obsah CD ROM