Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Piotr Adam Tabor
Nr albumu: 214569

Projekt i implementacja protokołu sieciowego dla semistrukturalnej bazy danych

Praca magisterska
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem
dra hab. Krzysztofa Stencla

Maj 2008

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora (autorów) pracy

Streszczenie

Poniższa praca opisuje projekt polegający na wymianie protokołu sieciowego w semistrukturalnej bazie danych LoXiM. Zawiera dokumentację poszczególnych kroków tego zagadnienia: analizę i wskazanie wad architektonicznych i implementacyjnych w zastanym protokole, projekt nowego rozwiązania, a także implementację generatora kodów źródłowych protokołów sieciowych pewnej klasy na podstawie danego pliku XML dla najbardziej popularnych języków programowania - C++ i Javy.

Słowa kluczowe

Protokół sieciowy, LoXiM, SBQL, SBA, bazy danych, podejście stosowe, LDAP, generowanie kodu, ASN.1

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

C. Computer Systems Organization
C.2 Computer-communication networks
C.2.2 Network Protocols
Applications (SMTP, FTP, etc.)

Tytuł pracy w języku angielskim

Design and development of network protocol for semistructured DBMS

Spis treści

Rozdział 1

Wprowadzenie

Praca to opisuje przebieg projektu polegającego na wyspecyfikowaniu i zaimplementowaniu protokołu sieciowego dla bazy danych LoXiM, która to baza powstaje pod kierownictwem dr hab. Krzysztofa Stencla na Wydziale Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego.

Dokument ten omawia zagadnienie protokołów sieciowych w zakresie warstwy górnej modelu OSI (Open System Interconnection) [OSI], czyli:

- 5. Warstwy sesji
- 6. Warstwy prezentacji
- 7. Warstwy aplikacji

Zakres ten jest tożsamy z zakresem „Warstwy aplikacji" w modelu DoD (Department of Defense) [DoD].

Praca ta - jako całość - toczyła się dość długo. Pierwsze jej elementy powstały na jesieni roku 2006, ale mimo wszystko udało się utrzymać zgodność finalnego dzieła z aktualnymi potrzebami, a nawet zaimplementowany protokół przewiduje obsługę wielu funkcjonalności i zastosowań nieobecnych jeszcze w systemie LoXiM w momencie oddania pracy. W sekcji 1.2 znajdują się informacje o zakresie zrealizowanego projektu.

W trakcie przebiegu tej pracy powstało kilka dokumentów, które dobrze spełniają swoją rolę jako dzieła oddzielne - skierowane do czytelnika zainteresowanego szczególnymi aspektami tej pracy i z tego względu zamieszczam je jako dodatki do tego dokumentu. Opis tych elementów składowych znajduje się w sekcji 1.1.

Bezpośrednio w tym dokumencie chcę się skupić na kwestii przebiegu tej pracy, uzasadnieniu podjętych istotnych decyzji projektowych i rozważeniu trudności z którymi się spotkałem. Następne rozdziały poświęcam zatem rozważaniom dotyczącym poszczególnych faz projektu:

Analizie zastanego protokołu w bazie LoXiM - rozdział: 2

Projekt nowego protokołu dla bazy LoXiM - rozdział: 3

Implementacji nowego protokołu dla bazy LoXiM - rozdział: 4

Implementacja generatora protokołów dla bazy LoXiM - rozdział: 5

Przeprowadzone testy - rozdział: 6

Dokument ten został więc zorganizowany niemal chronologicznie, a zarazem zgodnie z przebiegiem prac. W niektórych tylko miejscach pozwoliłem sobie zawrzeć wnioski, które wynikły z mojego późniejszego doświadczenia, ale które tematycznie powinny zostać uwzględnione na danym etapie realizacji projektu.

Całość treści właściwej zamyka podsumowanie w rozdziale: 7.

W dodatkach - oprócz omówionych w następnym rozdziale dokumentów - znajduje się spis treści załączonej płyty CD (dodatek E).

1.1. Elementy składowe pracy magisterskiej

Oprócz poniższego dokumentu w skład pracy zostały włączone następujące dokumenty:

1.1.1. „Analiza zastanego protokołu sieciowego w bazie danych LoXiM"

Dodatek A

Dokument opisuje protokół sieciowy jaki zastałem w bazie danych LoXiM w październiku 2006 roku. Dokument był pisany w kontekście rozpoznania protokołu na potrzeby stworzenia sterownika JDBC jego używającego — co było moim pierwotnym zamierzeniem. Krytyka rozwiązania zawarta w tym dokumencie stała się przyczyną podjęcia decyzji o wymianie protokołu w bazie LoXiM na nowy.

1.1.2. „Protokół komunikacyjny dla bazy danych LoXiM - wersja 2.0"

Dodatek B

Dokument ten realizuje dwa istotne cele:

- Stanowił swojego rodzaju „zamówienie", czyli przedstawiał kontrakt jaki protokół będzie realizował, co ułatwiało rozmowy z autorami innych modułów systemu LoXiM oraz kierownikiem projektu i umożliwiało wykrycie braków funkcjonalnych bądź zagrożeń.
- Aktualnie stanowi on dokumentację obecnego protokołu w bazie danych LoXiM. Jest dokumentem, który każda osoba chcąca napisać narzędzie bezpośrednio komunikujące się z systemem LoXiM musi przeczytać i dokument ten powinien odpowiedzieć na wszelkie pytania dotyczące tego interfejsu.

1.1.3. „Analiza możliwości wykorzystania protokołu LDAP dla SBQL DB"

Dodatek C

Dokument ten rozważa kwestie możliwości wykorzystania protokołu LDAP do komunikacji z bazą danych opartą o język SBQL. Zainspirowany został pozornie podobnym modelem danych i wykazuje poważne trudności (mimo licznych podobieństw) w integracji obu rozwiązań.

1.1.4. „ProtoGen 1.0 - dokumentacja generatora protokołów sieciowych"

Dodatek D

Tekst ten stanowi pełną dokumentację narzędzia, które umożliwia wygenerowanie na podstawie zadanego deskryptora protokołu w formacie XML, jego implementację w wybranym języku programowania. Dokument omawia zarówno sposób użycia tego narzędzia jak i porusza kwestie jego wewnętrznej architektury oraz możliwości dalszej rozbudowy.

1.2. Cel

Pierwotnym zagadnieniem, którym chciałem się zająć w ramach pracy magisterskiej, było zbudowanie odpowiednika ORM (Object-Relational mapping) dla bazy LoXiM w Javie. Okazało się jednak, że baza danych LoXiM nie posiada sterownika JDBC, który by umożliwiał zastosowanie standardowych dla Javy metod łączności z bazami danych. Dlatego zainteresowałem się stworzeniem sterownika JDBC dla LoXiM'a. Niestety przeprowadzona analiza zastanego protokołu (patrz: 2) wykazała, że obecny protokół jest całkowicie nieużyteczny. W związku z tym, celem tej pracy stała się wymiana protokołu w systemie LoXiM na istotnie lepszy. Rozumiemy przez to:

- Uzyskanie stabilnego, bezpiecznego protokołu- umożliwiającego pełne wykorzystanie obecnych i potencjalnie przyszłych możliwości systemu LoXiM- autentykacji, przesyłania zapytań i uzyskiwania złożonych odpowiedzi, przerywania zapytań w trakcie ich wykonania,a także konfiguracji sesji z bazą danych.
- Uzyskanie protokołu potrafiącego pracować pomiędzy maszynami o różnych architekturach sprzętowych i programowych.
- Uzyskanie efektywnego (pod względem wykorzystania sieci i CPU) i łatwo rozszerzalnego protokołu.
- Moduły protokołu powinny stanowić wygodny i spójny interfejs - możliwie zgodny z dobrą praktyką programowania w języku dla którego zostały przygotowane. W szczególności w językach ze statyczną kontrolą typów pakiety danych powinny jej podlegać.
- Przygotowanie do implementacji sterownika JDBC w oparciu o ten protokół.

W trakcie prac nad systemem stało się oczywiste jeszcze jedno wymaganie - potrzebna jest implementacja tego protokołu w wielu językach programowania:

C++ - ze względu na to, że LoXiM jest napisany w języku C++

Java - ze względu na to, że sterownik JDBC musi zostać napisany w Javie

.NET (C#) - ze względu na to, że w maju 2007 powstała istotna część serwera LoXiM zaimplementowana w języku C#

itd.

Realizacja wsparcia dla wielu języków stała się przyczyną stworzenia generatora protokołów (patrz: 5).

Rozdział 2

Analiza zastanego protokołu

W październiku 2006 roku - kiedy przystępowałem do pracy z systemem LoXiM - nie istniała żadna dokumentacja, ani opis używanego przez ten system protokołu sieciowego. Były więc koniecznością - na podstawie kodu źródłowego - przeprowadzenie audytu tego jedynego (pomijając standardowe wyjście) interfejsu komunikacyjnego bazy danych ze światem zewnętrznym.

Opis ten został zawarty w dokumencie „Analiza zastanego protokołu sieciowego w bazie danych LoXiM" (dodatek A).

2.1. Wnioski z przeprowadzonej analizy

Przytoczę w tym dokumencie wnioski, które wypłynęły z przeprowadzonego przeze mnie audytu:

- Protokół ten nie będzie działał pomiędzy komputerami różniącymi się architekturą lub trybem kompilacji (big endian - little endian), (8, 16, 32, 64 bity).
- Brak określonych z góry rozmiarów pakietów i ich elementów - co utrudnia stronie odbierającej alokowanie buforów o odpowiednich rozmiarach.
- Brak możliwości przerwania zleconego zapytania w trakcie jego wykonania
- Każdy pakiet ma inną konstrukcję i sposób zapisywania treści.
- Brak negocjacji wstępnych przy nawiązywaniu połączenia (logowanie, porównanie wersji, synchronizacja czasu, strona kodowa znaków w łańcuchach)
- Generalny brak konsekwencji.

  Czasami kod korzysta z funkcji zawartych w pliku TCPIP.cpp,a innym razem realizuje całkowicie analogiczną operacje bezpośrednio.

  Dla przykładu - pola tekstowe są przesyłane na jeden z 3 sposobów:

  – Jako ciągi bajtów, poprzedzone liczbą świadczącą o ich długości (wliczając znak \000 na końcu).
  – Jako ciągi bajtów, poprzedzone liczbą świadczącą o ich długości (nie wliczając znak \000 na końcu).
  – Jako ciągi bajtów - z założeniem, że znak \000 kończy przesyłany napis.

- Brak organizacji wewnętrznej pakietów umożliwiającej wygodną rozbudowę protokołu (o nowe dane w poszczególnych pakietach)
- Obsługa maksymalnie maxint+1 obiektów (czyli np. 32768 na architekturach 16-bitowych).
- Protokół nie przewiduje możliwości przesyłania liczb całkowitych ujemnych.
- Duplikacja kodu - wiele fragmentów kodu jest skopiowanych (copy-paste) z innego miejsca. Można by tego uniknąć organizując kod w odpowiedni sposób.
- Istnieją trzy typy związane z przesyłaniem danych logicznych: Result::BOOLTRUE, Result::BOOLFALSE, Result::BOOL - każdy serializowalny z innym identyfikatorem. Prawdopodobnie któryś z nich powstał przez pomyłkę.

W wyniku tej krytyki została podjęta decyzja o zaprojektowaniu i wykonaniu nowego - pozbawionego powyższych wad - protokołu komunikacyjnego dla LoXiM'a.

Rozdział 3

Projekt nowego protokołu

3.1. Wybór klasy protokołu

Kluczową decyzją przy projektowaniu nowego protokołu komunikacyjnego, było podjęcie decyzji o jego typie. Poniżej spróbujemy przejrzeć dostępne możliwości i rozważyć ich wady i zalety:

3.1.1. Klasyczne protokoły tekstowe

Przykładami klasycznych protokołów tekstowych są protokoły: HTTP, FTP, NNTP, POP3, SMTP. Są to protokoły w których wszystkie polecenia i odpowiedzi są zapisywane w postaci tekstowych komend. Są one charakterystyczne dla wczesnego etapu rozwoju komunikacji w sieciach komputerowych, głównie ze względu na poniższe zalety:

- Możliwość bezpośredniej komunikacji człowieka z urządzeniem przy pomocy tekstowego protokołu (wystarczy narzędzie typu „netcat")
- Brak problemów z wymianą danych pomiędzy urządzeniami różnego typu i o różnych architekturach (big/little endian, długość słowa procesora). Przeważnie wymagane jest jedynie aby oba urządzenia honorowały standard ASCII w zakresie znaków o kodach $0 - 127$).
- Możliwość bezpośredniego logowania komunikacji na drukarkę.
- Duża łatwość szukania błędów.

Niestety posiadają one także wiele wad:

- Wysoki narzut na transmisję danych (w kategoriach rozmiaru przesłanych danych). Np. wysłanie liczby 17-to cyfrowej wymaga w protokole binarnym 8 bajtów, a w protokole tekstowym 17 bajtów.
- Konieczność parsowania i budowania komunikatów tekstowych (mało wygodne dla programisty oraz wprowadzające istotny narzut obliczeniowy).
- Trudność przesyłania danych o złożonej strukturze.

3.1.2. Protokoły oparte na XML

Przykładem takiego protokołu jest protokół XMPP (Extensible Messaging and Presence Protocol) wykorzystywane przez komunikatory internetowe (jabber, gtalk).

Protokoły tego typu posiadają większość zalet protokołów tekstowych (patrz: 3.1.1). Jedynie czytelność i możliwość bezpośredniej obsługi protokołu przez człowieka uległa obniżeniu. Ponadto do zalet należy doliczyć:

- mnogość narzędzi umożliwiających np. automatyczną kontrolę poprawności komunikatów (XML Schema) lub przekształcanie komunikatów z jednej postaci na drugą (XSLT).
- obecność wielu „standardów" wymiany treści określonego typu
- obsługę wielu kodowań znaków.
- możliwość wymiany danych o złożonej strukturze,

W wadach istotne pozostają dwie charakterystyczne dla protokołów tekstowych:

- Wysoki narzut (w kategoriach wykorzystania sieci) na transmisję danych.
- Konieczność parsowania komunikatów i narzut obliczeniowy z tym związany (istotnie mniejszy niż w przypadku klasycznej komunikacji tekstowej).

Protokoły XML oparte na RPC (Remote Procedure Call) - np. XML-RPC, SOAP, Web-Services

Należy w szczególny sposób wyróżnić zestandaryzowaną klasą protokołów sieciowych związanych ze zdalnym wywoływaniem procedur i stanowiących obecnie powszechnie używany standard w komunikacji sieciowej pomiędzy różnymi systemami tzw. Web-Services.

Wszystkie one wyróżniają w komunikacji stronę zadającą zapytanie (request) i stronę udzielającą odpowiedzi (response). Strona udzielająca odpowiedzi udostępnia metodę (zawierającą potencjalnie złożone - obiektowe - parametry), która poprzez odpowiednio sformatowane zapytanie jest uruchamiana. Wartość wynikowa wykonanej metody jest kodowana w postaci XML'a i zwracana do strony pytającej.

Zaletą tych protokołów jest bez wątpienia:

- jeszcze większe zestandaryzowanie (WSDL)
- duża ilość narzędzi wspierających użytkowanie tych protokołów
- Istnienie generatorów kodu, które na podstawie opisu takiego protokołu (WSDL) generują dla wielu języków programowania kod potrzebny do komunikacji.
- Używanie portu 80 i warstwy transportu opartej o HTTP, co umożliwia uniknięcie problemów związanych z działaniem zapór sieciowych i szczególnych polityk bezpieczeństwa.

Niestety wprowadzają one też pewne istotne wady:

- Są bezpołączeniowe - wymagają nawiązania połączenia i przeprowadzenia procesu autoryzacji przy każdym wywołaniu - co ma bardzo negatywny wpływ na wydajność i bezpieczeństwo, a także może spowodować ograniczenie komunikacji przez urządzenia sieciowe (zbyt wiele żądań w zadanym okresie czasu).

- Wyróżniają stronę zadającą pytanie i na nią odpowiadającą. Komunikacja dwustronna wymaga tego, aby obie strony umiały zainicjalizować połączenie - co w przypadku sieci opartych np. o maskaradę IP może być trudne lub nawet niemożliwe.

3.1.3. Protokoły binarne oparte na RPC

Istnieje wiele rozwiązań zdalnego wywoływania procedur pomiędzy komponentami w sieciach komputerowych wykorzystujących komunikację binarną. Mechanizmem, który powinien umożliwić stworzenie takiego rozwiązania jest standard CORBA (Common Object Request Broker Architecture). Teoretycznie powinien on umożliwić wygenerowanie na podstawie zadanego IDL'a (Interface Description Language) implementacji protokołu dla wielu różnych języków programowania. Niestety rzeczywistość pokazuje, że nie istnieje dobra - niekomercyjna - implementacja standardu CORBA (http://www.puder.org/corba/matrix/).

Pozostałe protokoły - takie jak RMI (Remote Method Invocation) oraz AMF (Action Message Format) są związane z konkretnymi językami programowania (w tym przypadku odpowiednio Java i ActionScript).

3.1.4. Protokoły oparte na ASN.1 (Abstract Syntax Notation One)

ASN.1 jest standardem służącym do opisu metod kodowania, dekodowania i przesyłania danych. Jest to obecnie standard ISO/IEC 8824. Pozwala on zdefiniować za pomocą sformalizowanego opisu składnie pól w komunikatach, a następnie wygenerować kod seralizujący i deserializujący te pakiety.

Standard ASN.1 nie definiuje bezpośrednio binarnego formatu przesyłanych komunikatów. Mogą być one serializowalne według jednej z zaproponowanych zasad (encoding rules). W szczególności istnieją trzy najpopularniejsze możliwości w tej kwestii:

BER - (Basic encoding rules) - zapamiętuje każde pole w postaci binarnej jako trójkę: znacznik, długość, wartość.

PER - (Packed encoding rules) - podobnie jak BER, ale metoda bardzo zwraca uwagę na efektywność pod względem rozmiaru pakietów.

XER (XML encoding rules) - komunikaty są przesyłane w postaci paczek XML.

Można ten standard porównać do zaprezentowanego w ramach tej pracy generatora protokołów.

Protokół LDAP (Lightweight Directory Access Protocol)

Szczególnym przypadkiem protokołu opartego na standardzie ASN.1 jest protokół LDAP (Lightweight Directory Access Protocol) służący do wymiany danych z usługami katalogowym (Directory Services). Ze względu na podobne zastosowanie (uzyskiwanie dostępu do bazy danych o hierarchicznej strukturze) wydała mi się warta głębszej analizy kwestia rozważenia możliwości wykorzystania tego protokołu (z ewentualnymi rozszerzeniami) - jako protokołu do bazy danych LoXiM.

Problematykę tę szczegółowo omawia załączony do tej pracy dokument mojego autorstwa pt.: „Analiza możliwości wykorzystania protokołu LDAP dla SBQL DB" (patrz dodatek C).

3.1.5. Protokoły dedykowane

Są to protokoły binarne specjalnie zaprojektowane do konkretnych rozwiązań.

3.1.6. Wnioski

Z protokołów tekstowych najlepiej nadawałby się do omawianego zastosowania protokół oparty na XML, ale nie będący usługą „Web-Service" (zdyskwalifikowany ze względu na bezpołączeniowość).

Jednak wysoki narzut związany zarówno z transmisją jak i parsowaniem danych przeważył decyzję na rzecz protokołów binarnych (wyzwaniem postawionym bazie danych LoXiM jest udowodnienie, że obiektowe/semistrukturalne bazy danych mogą konkurować pod względem wydajności z bazami relacyjnymi, więc nie chcieliśmy wprowadzić wąskiego gardła na poziomie tego komponentu systemu).

Dysponując obecną wiedzą, dla systemu LoXiM zaleciłbym z pewnością protokół zbudowany w oparciu o standard ASN.1 (patrz: 3.1.4). Pozwoliłoby to pozostać w pełni zgodnym ze standardami ISO, a także uniknąć istotnej części implementacji - posługując się którymś z generatorów kodu dla standardu ASN.1.

Protokół ten zostałby zapewne oparty na kanwie protokołu LDAP (analogiczna konstrukcja paczek, zgodna autoryzacja), ale do przesyłania zapytań i odczytywania ich wyników zaproponowałbym własne paczki - semantycznie zgodne z tymi zaproponowanymi w sekcjach B.5.5 i B.5.6.

Podejmując tę decyzję projektową w grudniu 2006 roku, odrzuciłem protokół tekstowy ze względu na zbyt niską wydajność, a także użycie CORBY ze względu na niesatysfakcjonującą jakość bezpłatnych produktów i w ten sposób zdecydowałem się zaprojektować protokół dedykowany.

3.2. Projekt dedykowanego protokołu

Projekt, a tym samym dokumentacja dedykowanego protokołu sieciowego dla bazy danych LoXiM znajduje się w załączniku do tej pracy zatytułowanym: „Protokół komunikacyjny dla bazy danych LoXiM - wersja 2.0". Dokument ten szczegółowo omawia kwestie takie jak:

- Format binarny poszczególnych pakietów
- Dozwolone sekwencje wymiany pakietów
- Metody autoryzacji
- Kwestie bezpieczeństwa w sieci
- Podział protokołu na warstwy logiczne
- Problemy związane z danymi regionalnymi, takimi jak strefy czasowe, metody porównywanie napisów.

Rozdział 4

Implementacja protokołu

Po przedstawieniu „Projektu protokołu sieciowego dla bazy danych LoXiM" i omówieniu go na seminarium - zostały wprowadzone do niego niewielkie zmiany i w tej postaci został skierowany do realizacji.

Jako, że system LoXiM jest napisany w języku C++, kluczowa była implementacja protokołu w tym właśnie języku programowania. Protokół udało się zaimplementować dokonując tylko kosmetycznych zmian w stosunku do pierwotnego projektu.

Istotną częścią tej implementacji było stworzenie wygodnego - obiektowego - API do obsługi strumieni i gniazd sieciowych. Implementując je wzorowałem się w istotnym stopniu na tym udostępnianym przez klasy w języku Java takie jak: InputStream, OutputStream, Socket, ClientSocket i ServerSocket.

Opis tej implementacji protokołu można znaleźć w rozdziale: 3.2 „Wygenerowany kod dla języka C++" dokumentu „ProtoGen 1.0 - dokumentacja generatora protokołów sieciowych" (dodatek D), ze względu na to, że ten kod został wykorzystany jako baza dla generowanego kodu do języka C++.

Rozdział 5

Generator implementacji protokołów

5.1. Geneza

Implementując protokół w C++, stwierdziłem, że ponad 70% czasu zajęło mi dość mechaniczne tworzenie kodu poszczególnych pakietów, a pozostałe 30% czasu powstawał kod, który był niemal niezależny od protokołu z którym miałem do czynienia.

Ponadto - w maju 2007 roku - gdy skończyłem implementacje protokołu w języku C++ - pojawił się zaczątek implementacji serwera LoXiM w języku C# na platformie Microsoft .NET. Widząc więc potrzebę stworzenia implementacji tego protokołu w dwóch kolejnych językach programowania (C# i Java na potrzeby sterownika JDBC), a także konieczność utrzymania tych 3 implementacji spójnymi przy wszelkich modyfikacjach, doszedłem do wniosku, że nieodzowne wydaje się stworzenie generatora implementacji protokołu wedle zadanego jego opisu.

5.2. Opis

W dokumencie „ProtoGen 1.0 - dokumentacja generatora protokołów sieciowych" (dodatek D) znajduje się dokumentacja zarówno użytkowa jak i programistyczna tego narzędzia. Program ten (napisany w Javie) zawiera obecnie moduły generujące kod do języka C++ i do języka Java, ale posiada możliwość łatwego rozbudowania o producentów kodu do kolejnych języków programowania.

Zaimplementowany generator wraz z plikami źródłowymi znajduje się na załączonej do pracy płycie CD-ROM w katalogu "/protogen".

Jako ciekawostkę chciałbym zwrócić uwagę na różnicę w ilości kodu źródłowego, który był potrzebny do napisania modułów generatora (o całkowicie zgodnej funkcjonalności) :

Moduł generujący kod do C++ 84 785 bajtów kodu źródłowego

Moduł generujący kod do Javy 56 091 bajtów kodu źródłowego

Czyli kod generujący do języka C++ jest o 50% dłuższy od kodu generującego do języka Java. Proporcja ta wynika głównie z konieczności generowania plików nagłówkowych dla języka C++.

5.3. Wygenerowany kod dla LoXiM'a

W katalogu "/protogen/example" załączonej płyty został umieszczony pełen zbiór plików potrzebnych do wygenerowania kompletnego protokołu dla C++ i Javy. Oprócz pliku xml deskryptora stanowią go ręcznie przygotowane pliki dwóch paczek: CollectPackage i Q_c_executePackage, których logika była na tyle skomplikowana, że generator implementacji protokołu „ProtoGen 1.0" jej obecnie nie wspiera.

Aby ręcznie przeprowadzić proces generowania kodu, najlepiej jest:

1. skopiować na dysk lokalny cały katalog /protogen z załączonej płyty CD (podkatalog /protogen/src jest zbędny).

2. upewnić się, że posiadamy prawa zapisu do skopiowanego podkatalogu ./protogen/example. Ewentualnie nadać odpowiednie przywileje.

3. uruchomić program ./protogen/example/run.sh

W katalogu ./protogen/example/result-cpp i ./protogen/example/result-java powinny zostać wygenerowane implementacje protokołu.

Porównanie ilości wygenerowanego kodu (klasy paczek i typy wyliczeniowe) dla obu protokołów sieciowych pokazują podobny wynik:

Wygenerowany kod dla C++ 131 278 bajtów kodu źródłowego

Wygenerowany kod dla Javy 120 591 bajtów kodu źródłowego

Oprócz tego - dla obu języków - zostały wygenerowane około 850KB pliki zawierające testowe instancje pakietów (patrz: 6).

Gotowe implementacje protokołu dla LoXiM'a są także załączone na płycie w katalogu /loxim_protocol.

Rozdział 6

Przeprowadzone testy

6.1. Metoda i narzędzia

Ręcznie napisana implementacja protokołu w języku C++ zawierała przykładowe scenariusze testowe zaadresowane zarówno dla strony będącej serwerem jak i dla strony będącej klientem protokołu LoXiM'a.

Idea ta została także przeniesiona do generatora protokołów „ProtoGen 1.0" w którym to scenariusze testowe są przygotowywane automatycznie i są tożsame pomiędzy różnymi docelowymi językami programowania.

Dla deskryptora protokołu LoXiM'a generator stworzył opisy 3197 przykładowych paczek. Dla języka C++ zostały one zawarte w pliku:
/loxim_protocol/cpp/protocol/tests/TestPackagesFactory.cpp
a dla języka Java w pliku:
/loxim_protocol/java/src/test/java/pl/edu/mimuw/loxim/protocol/tests/TestPackagesFactory.java.

Dla każdego z języków programowania jest tworzony program: TestRunnerRec, który uruchomiony z parametrem będącym numerem portu - tworzy instancję serwera oczekującego na połączenie na wybranym porcie i sprawdzającego zgodności otrzymanych pakietów z zaplanowanym scenariuszem, oraz program TestRunnerSender, który uruchomiony z dwoma parametrami: adresem hosta docelowego i numerem portu na którym nasłuchuje tam serwer - podłącza się do wybranego serwera i wysyła do niego paczki według zadanego scenariusza.

6.2. Sprawdzone przypadki

Testowe maszyny:

1. L64 - Linux 2.6.20, Intel Dual Core - 64 bity, Little-endian
2. L32 - Linux 2.6.20, Intel Dual Core - 32 bity, Little-endian
3. B32 - AIX, Power PC - 32 bity, Big-endian

Sprawdzono, że system przechodzi testy w następujących scenariuszach

1. L64 Java $\Longleftrightarrow$ L64 Java
2. L64 C++ $\Longleftrightarrow$ L64 C++
3. L64 C++ $\Longleftrightarrow$ L32 C++

4. L32 C++ $\Longleftrightarrow$ L32 C++
5. L32 C++ $\Longleftrightarrow$ L32 Java
6. L64 C++ $\Longleftrightarrow$ B32 C++
7. B32 C++ $\Longleftrightarrow$ B32 C++
8. L64 Java $\Longleftrightarrow$ L64 C++

Rozdział 7

Podsumowanie

Uważam, że udało się zrealizować postawione w pracy cele. Został stworzony wydajny, przenośny, dobrze udokumentowany i dostosowany do obecnych i przewidywanych przyszłych potrzeb LoXiM'a protokół sieciowy.

Powstał także uniwersalny generator protokołów sieciowych - którego pierwotnie planowany zakres pracy nie dotyczył.

W momencie oddawania tej pracy (maj 2008) toczą się dwie prace magisterskie w bardzo istotnym stopniu oparte na wynikach opisanych w tym dokumencie.

Praca Marka Dopiery - polegająca na re-implementacji modułu „Listener" serwera LoXiM - odpowiedzialnego za nawiązywanie połączeń z klientami i zarządzanie ich zleceniami.

Praca Adama Michalika - polegająca na implementacji sterownika JDBC dla bazy danych LoXiM - zainspirowana moimi pierwotnymi planami.

Uwagi autorów tych prac przyczyniły się do drobnych poprawek w przedstawionych modułach i uściślenia niejasnych kwestii w dokumentacjach.

Dodatek A

Analiza zastanego protokołu sieciowego w bazie danych LoXiM(stan na 2006-10-26)

A.1. Wstęp

Celem tego dokumentu jest udokumentowanie protokołu sieciowego, jaki jest wykorzystywany przez semistrukturalną bazę danych „LoXiM". Konieczność stworzenia tego dokumentu wynika z faktu, że bieżący protokół komunikacyjny nie jest w żaden sposób udokumentowany, co uniemożliwia stworzenie aplikacji integrujących się z tą semistrukturalną bazą danych.

W dokumencie pojawiają się adnotacje dotyczące użycia lub jego braku - funkcji „htonl". Jest to standardowa funkcja języka C, która służy do konwersji danej liczby z architektury lokalnej komputera na liczbę kodowaną w systemie Big-endian.

W poniższym dokumencie pojawiają się też rozmiary i offsety względem początku paczki zapisane w nawiasach zwykłych. Używałem ich do oznaczenia sytuacji prawdziwej w przypadku architektury 64 bitowej (w przeciwieństwie do domyślnie przyjętej w tym dokumencie architektury 32bitowej). Oczywiście występują analogiczne różnice dla pozostałych architektur - o innym rozmiarze słowa procesora.

A.2. Rodzaje pakietów

SimpleQueryPackage Przesłanie prostego (bezparametrowego zapytania)

ErrorPackage Odpowiedź serwera stwierdzająca wystąpienie błędu.

ParamQueryPackage Wysłanie zapytania, które może zawierać parametry.

StatementPackage Otrzymanie od serwera ID wysłanego zapytania.

ParamStatementPackage Wysłanie parametrów do już wysłanego zapytania na serwer.

SimpleResultPackage Odpowiedź od serwera z wynikami przetwarzania zapytania.

RemoteQueryPackage Przetwarzanie rozproszone. Serwer prosi inny serwer o wykonanie podzapytania.

RemoteResultPackage Przetwarzanie rozproszone. Serwer odpowiada wynikami zapytania na zapytanie.

A.3. Standardowy format pakietu

O pakiecie jest wiadomo tylko jedno: zaczyna się od jednobajtowej stałej świadczącej o typie pakietu.

Od - do	Wartość	Zawartość
0 ⟷ 0		Stała określająca typ pakietu
1 ⟷ n		Dane pakietu

A.4. Standardowy format struktury danych

A.4.1. Obiekt tekstowy = Result::STRING

Przesyła łańcuch tekstu.

Od - do	Wartość	Zawartość
0 ⟷ 3(7)	Result::STRING	Stała mówiąca o typie rozważanego obiektu
4 (8) ⟷ 7 (15)	n (unsigned long)	Długość stringa (zast. fun. „htonl")z wliczonym znakiem \000
8 (16) ⟷ n+7 (n+15)	unsigned long	Przesyłany łańcuch
n+7 (n+15) ⟷ n+7 (n+15)	\000	Znak końca łańcucha

A.4.2. Obiekt pusty = Result::VOID

Dane puste - NULL.

Od - do	Wartość	Zawartość
0 ⟷ 3(7)	Result::VOID	Stała mówiąca o typie rozważanego obiektu

A.4.3. Informacja o błędzie = Result::ERROR

Przysyła unsigned long int (co z liczbami ujemnymi !!!).

Od - do	Wartość	Zawartość
0 ⟷ 3(7)	Result::ERROR	Stała mówiąca o typie rozważanego obiektu
4(8) ⟷ 7(15)	unsigned long	Numer błędu (zast. fun. „htonl")

A.4.4. Liczba całkowite (dodatnia) = Result::INT

Przysyła unsigned long int (co z liczbami ujemnymi !!!).

Od - do	Wartość	Zawartość
0 ⟷ 3(7)	Result::INT	Stała mówiąca o typie rozważanego obiektu
4(8) ⟷ 7(15)	unsigned long	Dana liczba (zast. fun. „htonl")

A.4.5. Wartość rzeczywista = Result::DOUBLE

Przesyła liczbę rzeczywistą. Liczba jest nie poprawnie konwertowana za pomocą funkcji (zast. fun. „htonl")

Od - do	Wartość	Zawartość
0 ⟷ 3(7)	Result::INT	Stała mówiąca o typie rozważanego obiektu
4(8) ⟷ 11(15)	double	Dana liczba - Nie poprawne użycie funkcji „htonl” - każde 4 bajty oddzielnie ?!

A.4.6. Prawda = Result::BOOLTRUE

Typ prawdy.

Od - do	Wartość	Zawartość
0 ⟷ 3(7)	Result::BOOLTRUE	Stała mówiąca o typie rozważanego obiektu
4(8) ⟷ 7(15)	1	Stała mówiąca, że to prawda (sam typ jak widać nie wystarczył autorowi tego rozwiązania)

A.4.7. Fałsz = Result::BOOLFALSE

Typ fałszu

Od - do	Wartość	Zawartość
0 ⟷ 3(7)	Result::BOOLFALSE	Stała mówiąca o typie rozważanego obiektu
4(8) ⟷ 7(15)	0	Stała mówiąca, że to fałsz (sam typ jak widać nie wystarczył autorowi tego rozwiązania)

A.4.8. Typ logiczny = Result::BOOL

Typ wartości logicznej (logika dwuwartościowa)

Od - do	Wartość	Zawartość
0 ⟷ 3(7)	Result::BOOL	Stała mówiąca o typie rozważanego obiektu
4(8) ⟷ 7(15)	0 lub 1	Stała mówiąca,czy to fałsz(0), czy prawda(1)

A.4.9. Łącznik = Result::BINDER

Binder - dowiązanie pomiędzy nazwą obiektu a wartością związaną z tą nazwą.

Od - do	Wartość	Zawartość
0 ⟷ 3(7)	Result::STRING	Stała mówiąca o typie rozważanego obiektu
4 (8) ⟷ 7 (15)	n (unsigned long)	Długość stringa (zast. fun. „htonl”)z wliczonym znakiem \000
8 (16) ⟷ n+7 (n+15)	unsigned long	Przesyłany łańcuch
n+7 (n+15) ⟷ n+7 (n+15)	\000	Znak końca łańcucha
n+8 (n+16) ⟷ ...		Dane związane z tą nazwą. Patrz: Standardowy format struktury danych - rekurencja

A.4.10. Multizbiór = Result::BAG

Implementowane przez funkcję get/setResultCollection. Reprezentuje multizbiór obiektów.

Od - do	Wartość	Zawartość
$0 \longleftrightarrow 3(7)$	Result::BAG	Stała mówiąca o typie rozważanego obiektu
$4(8) \longleftrightarrow 7(15)$	unsigned int	Liczba mówiąca o liczbie obiektów w rozważanej kolekcji
$8(16) \longleftrightarrow \ldots$		Kolejne definicje poszczególnych obiektów (patrz: Standardowy format struktury danych - czyli rekurencja)

A.4.11. Sekwencja = Result::SEQUENCE

Tak samo jak BAG. (Różni się tylko identyfikatorem typu struktury danych w pierwszych 4 (8) bajtach: Result::SEQUENCE)

A.4.12. Struktura = Result::STRUCT

Tak samo jak BAG. (Różni się tylko identyfikatorem typu struktury danych w pierwszych 4 (8) bajtach: Result::STRUCT)

A.4.13. Referencja = Result::REFERENCE

Referencja.

Od - do	Wartość	Zawartość
$0 \longleftrightarrow 3(7)$	Result::REFERENCE	Stała mówiąca o typie rozważanego obiektu
$4(8) \longleftrightarrow 7(15)$	int	Liczba będąca logicznym ID obiektu. UWAGA: liczba ta obecnie nie jest poddawana konwersji funkcją „htonl”. . . !!!

A.4.14. Wynik = Result::RESULT

Typ nie obsługiwany przy serializacji i deserializacji - zwraca kod błędu -2.

A.5. Przegląd typów pakietów

A.5.1. SimpleQueryPackage

Pakiet tego typu służy do przesłania prostego (bezparametrowego zapytania).

Od - do	Wartość	Zawartość
0 ⟷ 0	0	Stała określająca typ pakietu
1 ⟷ n		Łańcuch określający zapytanie - nie może zawierać znaku \000
n+1 ⟷ n+1	\000	Znak końca string'a (\000) - a tym samym końca pakietu

A.5.2. ErrorPackage

Pakiet zawiera numer błędu, który wystąpił po stronie serwera.

Od - do	Wartość	Zawartość
0 ⟷ 0	5	Stała określająca typ pakietu
1(1) ⟷ 4(8)		Numer błędu - wielkości sizeof(int) - a więc zależny od architektury!!!

A.5.3. ParamQueryPackage

Wysłanie zapytania, które może zawierać parametry. W strukturze komunikat nie różni się niczym od SimpleQueryPackage (tzn. jedyną różnicą jest identyfikator typu pakietu).

Od - do	Wartość	Zawartość
0 ⟷ 0	1	Stała określająca typ pakietu
1 ⟷ n		Łańcych określający zapytanie - nie może zawierać znaku \000
n+1 ⟷ n+1	\000	Znak końca string'a (\000) - a tym samym końca pakietu

A.5.4. StatementPackage

Otrzymanie od serwera ID wysłanego zapytania.

Od - do	Wartość	Zawartość
0 ⟷ 0	2	Stała określająca typ pakietu
1 ⟷ 4 (8)	unsigned long	Numer nadany utworzonemu zapytaniu

A.5.5. ParamStatementPackage

Wysłanie parametrów do już wysłanego zapytania na serwer.

Od - do	Wartość	Zawartość
0 ⟷ 0	3	Stała określająca typ pakietu
1 ⟷ 4 (8)	N - unsigned long	Całkowita długość tego pakietu
5 (9) ⟷ 8 (16)	unsigned long	Numer ID zapytania do którego przesyłamy parametry
9 (17) ⟷ 12 (24)	C - unsigned long	Liczba parametrów przekazywanych do zapytania
13 (25) ⟷ N		C wystąpień bloku opisującego pojedynczy parametr

Blok opisujący pojedynczy parametr

Pojedynczy parametr jest opisany następującą konstrukcją:

Od - do	Wartość	Zawartość
0 ⟷ 3 (7)	n (unsigned long)	Długość nazwy parametru (ze znakiem pustym)
4 (8) ⟷ n+2 (n+6)	unsigned long	Nazwa parametru
n+3 (n+7) ⟷ n+3 (n+7)	\000	Znak końca łańcucha
n+4 (n+8) ⟷ ...		Definicja wartości (patrz: Standardowy format struktury danych)

A.5.6. SimpleResultPackage

Odpowiedź od serwera z wynikami przetwarzania zapytania.

Od - do	Wartość	Zawartość
0 ⟷ 0	4	Stała określająca typ pakietu
1 ⟷ ...		Definicja wartości (patrz: Standardowy format struktury danych)

A.5.7. RemoteQueryPackage

Przetwarzanie rozproszone. Serwer prosi inny serwer o wykonanie podzapytania.

Od - do	Wartość	Zawartość
0 ⟷ 0	6	Stała określająca typ pakietu
1 ⟷ 1		Informacja o dereferencji (0-nie,1-tak)
2 ⟷ N		Remote LogicalID

A.5.8. RemoteResultPackage

Przetwarzanie rozproszone. Serwer odpowiada wynikami zapytania na zapytanie.

Od - do	Wartość	Zawartość
0 ⟷ 0	7	Stała określająca typ pakietu
1 ⟷ ...		Definicja wartości (patrz: Standardowy format struktury danych)

A.6. Struktura plików

Wszystkie pliki bezpośrednio związane z przesyłaniem danych przez sieć zawarte są w katalogu /TCPProto

A.6.1. Tcp.h i Tcp.cpp

Są to pliki zawierające „teoretycznie" niezależne funkcje do obsługi nawiązywania połączeń sieciowych i wysyłania nimi danych, a także serializowania prostych typów danych. Niestety istotna część tych funkcji jest nieoprawna (nie uważne stosowanie funkcji htonl - zamieniającej kolejność bajtów pomiędzy systemami opartymi o Big-endian i Little-endian).

Także użyte są nadal typu int, long - których rozmiar jest zależny od architektury na której kompilujemy system. Ponadto klasy te są użyte tylko w części przypadków - w pozostałych odczyty, konwersje i zapisy robione są ręcznie.

A.6.2. Package.h

Plik Package.h zawiera deklarację typu abstrakcyjnego z którego dziedziczą wszystkie klasy do przesyłania danych przez sieć:

```
class Package {
    public:
    enum packageType {
        RESERVED        = -1,   //to force nondefault deserialization method
        SIMPLEQUERY = 0,    //String
        PARAMQUERY = 1,     //String with $var
        STATEMENT = 2,      //Parser tree
        PARAMSTATEMENT = 3,//stmtNr + map
        SIMPLERESULT = 4,   //Result
        ERRORRESULT = 5,    //Parse/execute error package
        REMOTEQUERY = 6,    //remote reference
        REMOTERESULT = 7    //environment section
    };
    virtual packageType getType()=0;

    //returns error code, message buffer and size of the buffer
    //it doesn't destroy the buffer
    //first byte of the buffer is the resultType
    virtual int serialize(char** buffer, int* size)=0;

    //returns error code, gets buffer and it's size
    //it destroys the buffer
    virtual int deserialize(char* buffer, int size)=0;
    virtual ~Package(){}
};
```

Plik ten definiuje zatem identyfikatory pakietów zapisywane w pierwszym bajcie. Ponadto wymaga by klasa utożsamiana z pakietem posiadała metodę umożliwiającą wczytanie pakietu z danej tablicy bajtów (deserialize) i zapisanie pakietu do danej tablicy bajtów (serialize).

Ponadto w tym pliku nagłówkowym znajdują się deklarację wszystkich pakietów.

A.7. Stwierdzone wady protokołu

- Protokół ten nie będzie działał pomiędzy komputerami różniącymi się architekturą lub trybem kompilacji (big endian - little endian), (8, 16, 32, 64 bity).
- Brak określonych z góry rozmiarów pakietów i ich elementów - co utrudnia stronie odbierającej alokowanie buforów o odpowiednich rozmiarach.
- Brak możliwości przerwania zleconego zapytania w trakcie jego wykonania
- Każdy pakiet ma inną konstrukcję i sposób zapisywania treści.
- Brak negocjacji wstępnych przy nawiązywaniu połączenia (logowanie, porównanie wersji, synchronizacja czasu, strona kodowa znaków w łańcuchach)
- Generalny brak konsekwencji.

  Czasami kod korzysta z funkcji zawartych w pliku TCPIP.cpp,a innym razem realizuje całkowicie analogiczną operacje bezpośrednio.

  Dla przykładu - pola tekstowe są przesyłane na jeden z 3 sposobów:

  - Jako ciągi bajtów, poprzedzone liczbą świadczącą o ich długości (wliczając znak \000 na końcu).
  - Jako ciągi bajtów, poprzedzone liczbą świadczącą o ich długości (nie wliczając znak \000 na końcu).
  - Jako ciągi bajtów - z założeniem, że znak \000 kończy przesyłany napis.
- Brak organizacji wewnętrznej pakietów umożliwiającej wygodną rozbudowę protokołu (o nowe dane w poszczególnych pakietach)
- Obsługa maksymalnie maxint+1 obiektów (czyli np. 32768 na architekturach 16-bitowych).
- Protokół nie przewiduje możliwości przesyłania liczb całkowitych ujemnych.
- Duplikacja kodu - wiele fragmentów kodu jest skopiowanych (copy-paste) z innego miejsca. Można by tego uniknąć organizując kod w odpowiedni sposób.
- Istnieją trzy typy związane z przesyłaniem danych logicznych: Result::BOOLTRUE, Result::BOOLFALSE, Result::BOOL - każdy serializowalny z innym identyfikatorem. Prawdopodobnie któryś z nich powstał przez pomyłkę.

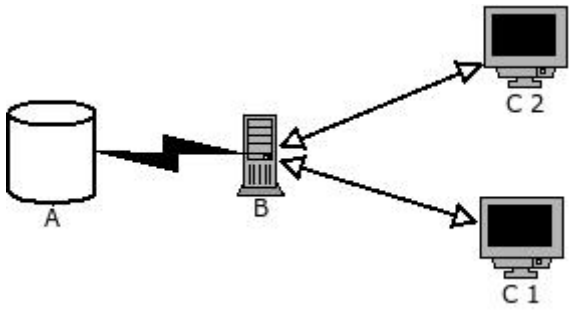
Dodatek B

Protokół komunikacyjny dla bazy danych LoXiM - wersja 2.0

B.1. Wstęp

Dokument ten opisuje protokół wymiany danych w systemie semistrukturalnej, obiektowej bazy danych opartej na stosowym języku zapytań (SBQL). Protokół służy do komunikacji pomiędzy aplikacją kliencką, a odpowiednim oprogramowaniem bazy danych — służącym do wykonywania zapytań i umożliwiającym autoryzację użytkowników.

Zatem rozpatrzmy hipotetyczną sytuację w której baza danych - rozumiana jako zbiór danych i mechanizm jej odczytów znajduje się na komputerze A, na komputerze B znajduje się proces przetwarzania zapytań, a na komputerze C znajduje się aplikacja chcąca korzystać z danych poprzez zadawanie zapytań. Opisany w poniższym dokumencie protokół służy do dwukierunkowej komunikacji pomiędzy aplikacjami działającymi na komputerach B i C, i nie nadaje się do komunikacji pomiędzy komputerami A i B.



Rysunek B.1: Diagram elementów systemu

Obecnie w systemie „LoXiM" część „A" (storage) i „B" (executor) są zintegrowane w jednej aplikacji.

Dokument ten został opisany na potrzeby projektu „LoXiM", ale z poszanowaniem warunków umowy licencyjnej (GPL) protokół może zostać wykorzystany w dowolnej bazie danych, której potrzeby spełnia. Rozwiązania zastosowane w tym protokole wydają się być na tyle ogólne, że z dużą pewnością mogą zaspokoić potrzeby wielu rozwiązań — także relacyjnych i nie opartych o SBQL'a.

Protokół ten stara się połączyć najlepsze cechy poniższych protokołów, jednocześnie skupiając się na wprowadzeniu możliwości pracy z danymi semistrukturalnymi:

Postgresql 10 — używanego przez PostgreSQL 8.2 [PGSQL]

TDS 8.0 — używanego przez Sybase i Microsoft SQL Server 7.5/2000 [TDS]

MySQL 3 — używanego przez serwer MySQL 5 [MYSQL]

B.1.1. Cele i założenia

Celem projektu opisanego przez ten dokument jest:

- Uzyskanie stabilnego. bezpiecznego protokołu, umożliwiającego pełne wykorzystanie obecnych możliwości systemu LoXiM.
- Uzyskanie protokołu potrafiącego pracować pomiędzy maszynami o różnych architekturach sprzętowych i programowych
- Uzyskanie efektywnego (pod względem wykorzystania sieci i CPU) i łatwo rozszerzalnego protokołu.
- Przygotowanie do implementacji sterownika JDBC w oparciu o ten protokół.

Przyjęto następujące założenia:

- Zakładamy, że komunikacja będzie się odbywała po odpornej na zakłócenia, błędy transmisji i zamianę kolejności przesyłanych danych warstwie transportowej (TCP/IP, Unix sockets, łącza nazwane Windows itp.).

  Zatem nie będziemy przeprowadzali własnej kontroli spójności przesłanych danych — pod względem sum kontrolnych itp. Jednak ze względów bezpieczeństwa kontrola logiczna będzie oczywiście przeprowadzana.

B.1.2. Wersjonowanie protokołu

Protokół komunikacji będzie wersjonowany przy pomocy dwóch numerów: głównego (major) i pomocniczego/pobocznego (minor). Poniższy dokument opisuje potoków w wersji 2.0, czyli numer główny 2, a pomocniczy 0. Przyjmuje się, że aplikacje posługujące się protokołem o tym samym numerze głównym są ze sobą zgodne - jedynie możliwości komunikacji są ograniczone do tych dostępnych w starszej wersji protokołu (mniejszy numer pomocniczy).

Zatem w obrębie tego samego numeru głównego można przewidzieć następujące typy zmian:

- Rozbudowanie formatu paczki poprzez dodanie na jej końcu nowych (nie obowiązkowych pól).
- Wprowadzenie nowych typów paczek — niekluczowych dla działania systemu i niemodyfikujących dotychczasowej semantyki operacji.

B.1.3. Licencjonowanie

Dokument ten — podobnie jak system LoXiM — jest licencjonowany na licencji GPL 2 (General Public License — wersja 2). W związku z tym protokół ten może być wykorzystany bez dodatkowej zgody autora w dowolnym systemie licencjonowanym na GPL.

B.2. Paczka — jednostka logiczna komunikacji

B.2.1. Paczka, a pakiet

Cała komunikacja będzie się opierała o przesyłanie paczek — spójnych ciągów danych o określonym formacie. Słowa „paczka" będziemy używali dla rozróżnienia i uniknięcia nieporozumień z pojęciem pakietu — który ma znaczenie na poziomie warstwy transportu (np. TCP/IP).

Zatem paczka to zbiór danych mających logiczne znaczenie w systemie LoXiM. Zupełnie nie wnikamy w to w jaki sposób paczki zostaną zorganizowane w pakiety i to zadanie pozostawiamy warstwie transportu. Jedyne co musimy zagwarantować, to fakt, że pojedyncza paczka jest spójnym obszarem danych w komunikacji.

W dokumencie tym będziemy także używali sformułowania „komunikat", które uznajemy za synonim słowa „paczka".

B.2.2. Budowa paczki

Każda paczka ma następujący schemat budowy:

Od - do	**Typ/Wartość**	**Zawartość**
$0 \rightarrow 0$	uint8	Stała mówiąca o typie paczki, a tym samym określająca format zawartych w nich danych
$1 \rightarrow 4$	uint32	n — stała określająca ilość danych właściwych zawartych w paczce – wyrażona w bajtach
$5 \rightarrow 4 + n$	patrz opis zależny od typu paczki	Dane właściwe paczki zgodne z formatem określonym poprzez typ paczki

Formalnie będziemy mówili, że paczka się składa z dwu-polowego nagłówka oraz ciała. Nagłówek wyznacza typ paczki i rozmiar danych właściwych w niej zawartych, a ciało — to dane interpretowane zależnie od typu paczki. Kluczową część tego dokumentu stanowi opis formatów poszczególnych paczek w zależności od ich typów.

Przyjmuje się, że rozmiar pojedynczej paczki nie może przekraczać 1MB (1'048'576 bajtów). Służy to uniknięciu sytuacji, w których odbywa się próba naruszenia bezpieczeństwa serwera poprzez przesłanie zbyt dużej paczki (doprowadzenie do wystąpienia błędu OutOfMemory), a także uniknięciu sytuacji w której traci się możliwość komunikacji asynchronicznej w skończonym czasie (np. wysłanie do serwera informacji o braku dalszego zainteresowania danymi). Wyżej wymieniona stała może być (a nawet powinna) konfigurowalna po stronie serwera.

B.2.3. Czemu przesyłamy długość paczki?

Przesłanie długości paczki na jej początku niesie za sobą następujące ułatwienia:

- Klient może zaalokować właściwą ilość danych w pamięci operacyjnej na przyjęcie całego komunikatu z góry — co ma pozytywny wpływ na wydajność i bezpieczeństwo (nie nadejdzie „nieskończenie" długi komunikat).

- Klient może zrezygnować z pobierana komunikatu (np. z powodu brak wystarczającej ilości pamięci, by go przyjąć) lub braku zainteresowania. Dzięki temu wie, ile bajtów musi zignorować bez ich analizy.

B.2.4. Mechanizmy rozszerzania protokołu

Protokół może być rozszerzany z zachowaniem zgodności wstecz poprzez dołączanie nowych pól na końcu danych paczki. Zatem strona odczytująca komunikat nie może zakładać, że po odczytaniu wszystkich pól w paczce o których wie, dotarła na koniec paczki. Możliwe, że strona ta przeczyta wszystkie pola i nie dotrze wcale do końca paczki, ponieważ w paczce występują dane dotyczące nowszej wersji protokołu. Zatem powinna ona zignorować odpowiednią ilość bajtów — wynikającą z długości danych w paczce, tak aby dotrzeć na początek następnego komunikatu.

Także możliwa jest odwrotna sytuacja. Strona wysyłająca posługuje się starszą wersją protokołu, więc wysyła komunikat nie zawierający pól wprowadzonych w nowszej wersji protokołu. Zatem strona odczytująca (która jest nowsza) powinna umieć właściwie zareagować na sytuację, gdy odczytano całą paczkę, a nie otrzymano nowych pól wprowadzonych w protokole. W tej sytuacji powinna przyjąć wartości domyślne dla tych pól.

Oczywiście odpowiednio duże zmiany i przeprojektowanie w protokole mogą wymagać stworzenia protokołu o kolejnym numerze głównym, czyli niezgodnego ze starszymi wersjami.

B.3. Podstawowe typy danych

B.3.1. Postanowienia ogólne

- Jeśli w sposób szczególny nie zaznaczono inaczej (a raczej nigdzie nie zaznaczono) wszystkie wartości zapisywane są w formacie Big-endian. W szczególności obejmuje to typy całkowitoliczbowe, rzeczywiste, oraz napisy w kodowaniu UTF-8 także stosują kolejność Big-endian.

B.3.2. Całkowitoliczbowe: uint8, sint8, uint16, sint16, int32, uint32, uint64, sint64

Pierwsza litera determinuje, czy mamy do czynienia z typem ze znakiem (u - unsigned), czy z typem bez znaku (s — signed). Liczba na końcu wyraża długość typu wyrażoną w bitach.

Typy są kodowane oczywiście w kolejności Big-endian.

varuint — Całkowitoliczbowy z kompresją (1,3,5 lub 9 bajtów)

Będzie to typ używany głównie do oznaczania długości stringów i ogólnie paczek.

Rozwiązanie techniczne zostało zaczerpnięte z protokołu serwera MySQL [MYSQL].

Idea jest taka, że krótkie stringi (< 250 znaków) będą się pojawiały najczęściej i chcemy mieć najmniejszy narzut na zapisanie długości (jedno bajtowy).

Zatem semantyka pierwszego bajtu jest następująca (w zależności od jego wartości)

0-249 Wartość ta jest jednocześnie wartością wynikową

250 Wartość jest null'em (zostawiamy dla umożliwienia stosowanie tego rozwiązania z systemami relacyjnymi)

251 Wartość ta poprzedza 2-bajtowe (uint16) pole w wartością właściwą

252 Wartość ta poprzedza 4-bajtowe (uint32) pole z wartością właściwą

253 Wartość ta poprzedza 8-bajtowe (uint64) pole z wartością właściwą. Nie należy jednak korzystać z wartości większych niż $2^{63} - 1$ (MAX_SINT64), ze względu na to, że w Javie nie są obsługiwane 8 bajtowe typy bez znaku.

B.3.3. string — Łańcuchy tekstu

Służy do przesyłania tekstów. Teksty te muszą być kodowane za pomocą UTF-8.

Format wartości typu string jest następujący: Najpierw idzie pole typu varuint — opisujące długość w bajtach następującego potem ciągu w UTF-8 (Big-endian).

B.3.4. sstring — Krótki łańcuch tekstu

Jest to tak naprawdę wariant typu string, ale ograniczony do łańcuchów nie dłuższych niż 249 bajtów. Czyli wtedy pole oznaczające długość ma jeden bajt. Jego wewnętrzna reprezentacja zupełnie się nie różni od typu string — został on wyróżniony formalnie — ze względu na uproszczenie notacji używanej przy opisach formatów poszczególnych paczek.

B.3.5. bytes — Dane binarne

Służy do przesyłania danych binarnych (nie koniecznie dużych — nie będziemy tego rozróżniali na poziomie protokołu).

Format wartości tego typu jest następujący: Najpierw zostaje przesłane pole typu varuint — opisujące długość w bajtach następującego potem ciągu bajtów.

B.3.6. Daty i czas

Kwestia stref czasowych

Można rozważyć dwa podejścia do przekazywania informacji o strefie czasowej:

- Przekazujemy tylko offset strefy czasowej względem GMT (czyli wartość od -14 do $+12$)
- Przekazujemy pełną informację o strefie czasowej. Wygląda na to, że nie istnieje standard ISO opisujący kodowanie dla wszystkich stref czasowych na świecie. Najlepszą bazą danych z informacjami o strefach czasowych jest baza TZ (znana też jako zoneinfo) (http://www.twinsun.com/tz/tz-link.htm). Koduje one informacja o strefie czasowej w postaci napisu: {Kontynent}/{Duże miasto} lub {Kontynent}/{Państwo}/{Duże Miszto}, np. „Africa/Porto-Novo".

  Zaletę przekazywania daty z tą pełną informacją, jest to, że dopiero na jej podstawie system informatyczny jest wstanie prawidłowo dodać pewną ilość czasu (np. 36 godzin) do danej daty w sytuacji, gdy w międzyczasie występuje zmiana czasu z letniego na zimowy lub odwrotnie.

Nie budzi wątpliwości, że dla uniknięcia problemów w aplikacjach o dużym zasięgu terytorialnym warto by było z datą zapisywać pełną informację.

Według bazy „TZ" w chwili obecnej na świecie jest używanych 398 różnych stref czasowych. Dobry standard pozwoliłby zakodować je za pomocą trzech liter, czyli — będąc rozrzutnym — 3 bajtów. Oszczędność 2 bajtów wydaje się być więc mało uzasadniona wobec ryzyka utraty poprawności numerycznej niektórych operacji, skoro stanowi ona wzrost długości całej zakodowanej daty z 9 do 11 bajtów, czyli o 22%.

Niestety za nieformalny standard w bazach danych (PostgreSQL, MySQL, Oracle, DB2) przyjęło się zapisywać jedynie informacje o przesunięciu względem GMT, co wynika ze złożenia dwóch problemów:

- Światowy standard formatu czasu ISO8601 [ISO8601] przewiduje tylko strefę czasową zapisaną w postaci przesunięcia względem GMT.
- Brak standardu (klasy ISO) kodowania stref czasowych, co pośrednio wynika z punktu powyższego.

Z tego powodu obecna wersja protokołu zapisuje strefy czas w „standardowy" sposób, czyli w postaci offsetów. Zalecamy jednak używanie pełnej informacji o strefie czasowej w następnych wersjach protokołu.

DATE

Sama data.

Format następujący:

Od - do	Typ/Wartość	Zawartość
$0 \rightarrow 1$	*Year* (sint16)	Rok
$2 \rightarrow 2$	*Month* (uint8)	Miesiąc (1-styczeń, 12-grudzień)
$3 \rightarrow 3$	*Day* (uint8)	Dzień

TIME

Sam czas.

Format następujący:

Od - do	Typ/Wartość	Zawartość
$0 \rightarrow 0$	*Hour* (uint8)	Godzina (0-23)
$1 \rightarrow 1$	*Minuts* (uint8)	Minuta (0-59)
$2 \rightarrow 2$	*Secs* (uint8)	Sekunda (0-59)
$3 \rightarrow 4$	*Milis* (sint16)	Milisekundy (0-999)

TIMETZ

Czas ze strefą czasową

Format następujący:

Od - do	Typ/Wartość	Zawartość
$0 \rightarrow 0$	*Hour* (uint8)	Rok
$1 \rightarrow 1$	*Minuts* (uint8)	Miesiąc (1-styczeń, 12-grudzień)
$2 \rightarrow 2$	*Secs* (uint8)	Dzień
$3 \rightarrow 4$	*Milis* (uint16)	Milisekundy
$5 \rightarrow 5$	*TZ* (sint8)	Strefa czasowa (-14 do +12)

DATETIME

Data i godzina bez strefy czasowej. Format zapisu to bezpośrednio po sobie występujące formaty pól typu DATE i TIME.

DATETIMETZ

Data i godzina ze strefą czasową. Format zapisu to bezpośrednio po sobie występujące formaty pól typu DATE i TIMETZ.

B.3.7. Logiczne

Pole typu bool niesie pewną wartość logiczną. W praktyce będzie reprezentowane jako liczba typu sint8 z następującymi wartościami:

0 Fałsz

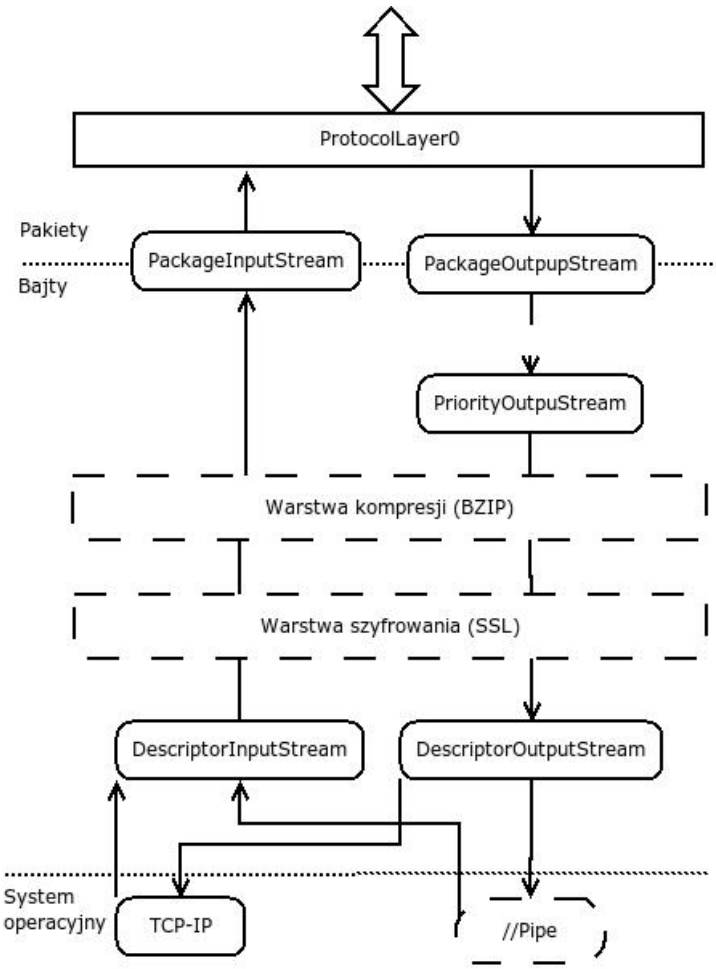
1 Prawda

B.3.8. Zmiennoprzecinkowe

DOUBLE

Podwójna precyzja, 8-bajty w kolejności Big-endian.

B.4. Podwarstwy



Rysunek B.2: Diagram warstw

Protokół w warstwie aplikacyjnej może chodzić w kilku podwarstwach. Elementy te umożliwiają szyfrowanie i kompresję w czasie działania protokołu.

W poniższych rozważaniach rozpatrzmy sytuację najbardziej złożoną — w której zarówno szyfrowania jak i kompresja jest dostępna.

B.4.1. Pisanie danych

Aplikacja chce wysłać paczkę do innej aplikacji:

1. Aplikacja umieszcza (zapisuje) wysyłaną paczkę do strumienia wyjściowego.

2. Strumień ten okazuje się być strumieniem kompresującym (ZLIB), który skompresowane dane zapisuje do swojego strumienia wyjściowego.

3. Strumień ten okazuje się być strumieniem szyfrującym (SSL), który dokonuje szyfrowania danych, a następnie umieszcza je w swoim strumieniu wyjściowym

4. Strumień ten okazuje się być strumieniem odpowiedniej warstwy transportu, która to warstwa przekazuje dane przez sieć.

B.4.2. Czytanie danych

Aplikacja chce odczytać paczkę pochodzą od innej aplikacji.

1. Aplikacja prosi o paczkę odpowiedni strumień wejściowy.
2. Strumień ten okazuje się być strumieniem dekompresującym (ZLIB), który prosi o dane swój strumień wejściowy.
3. Strumień ten okazuje się być strumieniem deszyfrującym (SSL), który prosi o dane swój strumień wejściowy.
4. Strumień ten okazuje się być strumieniem odpowiedniej warstwy transportu i odbiera on te dane z sieci, a następnie zwraca je.
5. Strumień deszyfrujący przetwarza dane i zwraca je
6. Strumień dekompresujący przetwarza dane i zwraca je
7. Aplikacja odczytuje dane i kompletuje je w całą paczkę.

B.4.3. Inicjalizacja tych podwarstw

Inicjalizacja odpowiednich podwarstw odbywa się tylko i wyłącznie w protokole wstępnym. Raz zainicjalizowanych podwarstw nie można wyłączyć.

B.5. Przepływ komunikatów

Sekcja ta opisuje przepływ komunikatów, a także format każdego z komunikatów. W protokole możemy wyróżnić kilka podprotokołów zależne od stanu w którym połączenie się znajduje. W szczególności bardzo sztywno należy odgrodzić protokół wstępny — w którym odbywa się negocjacja parametrów połączenia i autoryzacja z protokołem właściwym — w którym jest realizowana właściwa funkcjonalności systemu LoXiM, a zatem wykorzystywane są podprotokoły: przeprowadzania zapytań, przesyłania komunikatu asynchronicznego i kończenia połączenia.

B.5.1. Nazewnictwo paczek

Na potrzeby tego dokumentu przyjmujemy następujący schemat nazywania paczek:

{znacznik grupy komunikatów}-{znacznik strony, która wysyła tę paczkę}-{identyfikator znaczenia}

Gdzie znacznik grupy paczek może przyjąć następujące wartości:

W — Podprotokół wstępny

Q — Podprotokół przeprowadzania zapytań

V — Podprotokół przesyłania wartości

A — Komunikat asynchroniczny

S — Komunikaty standardowe (np. OK, ERROR)

Znacznikiem strony wysyłającej komunikat może być jedna z następujących możliwości:

C — Tylko klient wysyła tego typu komunikat

S — Tylko serwer wysyła tego typu komunikat

SC — Zarówno klient, jak i serwer mogą wysłać ten komunikat

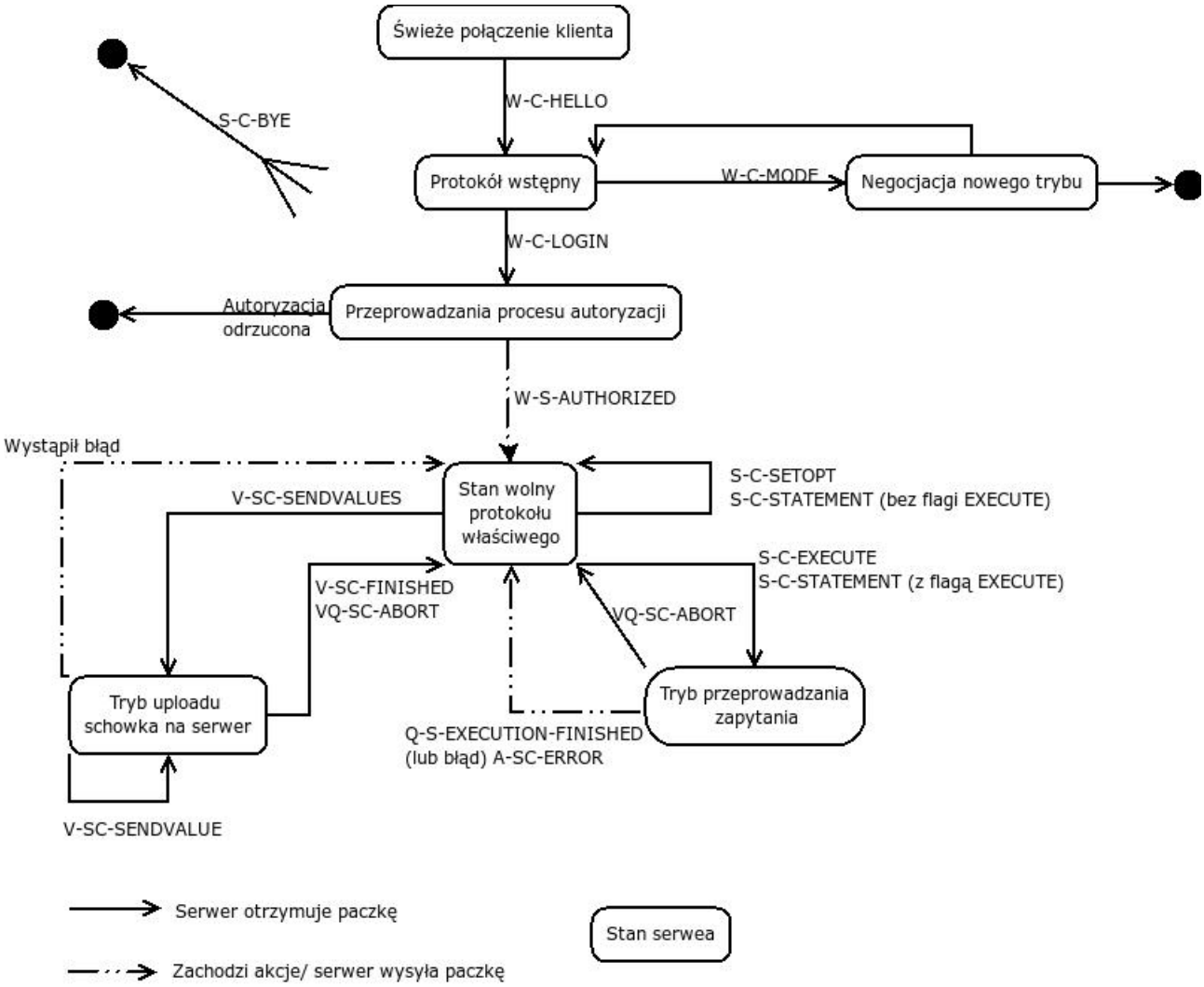
B.5.2. Metody opisu formatu paczki

Aby ułatwić zapoznawanie się z formatem paczek będziemy je prezentowali w tabeli o następującym formacie:

Od - do	**Typ/Wartość**	**Zawartość**
-5 → -5	10 (uint8)	Identyfikator paczki
-4 → -1	0 (uint32)	Długość właściwej zawartości paczki
0 → ...	wartość (typ)	Opis ogólny pole

Należy zwrócić uwagę, że będziemy prezentowali offsety względem segmentu danych.

B.5.3. Stany serwera



Rysunek B.3: Diagram przejść i stanów serwera

B.5.4. Protokół wstępny

Protokół wstępny służy do negocjacji parametrów połączenia (kto, do której bazy danych, czy z szyfrowaniem i kompresją, jakie są możliwości serwera i klienta), a także do przeprowadzenia autoryzacji użytkownika.

Ogólny schemat konwersacji jest następujący:

1. Klient nawiązuje połączenie (w przypadku TCP otwiera odpowiedni port na odpowiednim komputerze)
2. Serwer (o ile nie jest skonfigurowana polityka odrzucania połączeń z hosta klienta - raczej na zaporze sieciowej) przyjmuje (akceptuje) połączenie i nic nie wysyła.
3. Klient wysyła paczkę W-C-HELLO (patrz: B.5.4) — klient ujawnia, że chce rozmawiać z LoXiM'em po tym protokole.

4. Serwer odpowiada paczką W-S-HELLO (patrz: B.5.4) — serwer ujawnia swoje cechy i możliwości.

5. Klient ewentualnie (kilkukrotnie) wysyła paczkę W-C-MODE (patrz: B.5.4) — w którym prosi serwer o zmianę trybu (włączenie kompresji, szyfrowania). Odbywa się stosowna konwersacja związana z zamówionymi podwarstwami (jeśli serwer je obsługuje — np. SSL handshake). Serwer potwierdza przyjęcie zlecenia wykonania tej operacji S-SC-OK (patrz: B.5.8) lub zgłasza błąd S-SC-ERROR (patrz: B.5.8). Następnie może zostać przesłany zestaw paczek negocjujących nowy tryb. Następny komunikat jest transportowany już w nowym trybie.

6. Klient ewentualnie ustala pożądane cechy połączenia S-C-SETOPT (patrz: B.5.8)

7. Serwer potwierdza, bądź odrzuca ich przyjęcie.

8. Klient wybiera protokół logowania i informuje o nim serwer W-C-LOGIN (patrz: B.5.4). Serwer przeprowadza konwersację autoryzującą z klientem według jednej z metod. Podstawowe metody autentykacji opisane są w dalszej części tego dokumentu (patrz: B.7.6).

9. Ostatecznie serwer bądź informuje klienta o zatwierdzeniu autoryzacji W-S-AUTHORIZED (patrz: B.5.4) bądź odrzuca ją i zamyka połączenie S-SC-ERROR (patrz: B.5.8).

10. Protokół wstępny zostaje zakończony.

W-C-HELLO

Paczka służy nawiązaniu właściwego połączenia na poziomie logicznym pomiędzy klientem, a serwerem. Stanowi formę przedstawienia, dzięki której serwer nabywa przekonanie, że ma do czynienia z uczciwym klientem, a nie skanerem portów, a także przedstawia serwerowi dane, które mogą być przydatne — głównie do celów administracyjnych (śledzenie stanu serwera, diagnostyka).

Można wyróżnić następujące grupy pól w tym komunikacie

Cechy procesu klienta: PID, nazwa, wersja, hostname Służą one umożliwieniu śledzenia akcji mających miejsce aktualnie na serwerze oraz sporządzanie statystyk i logów dotyczących aktywności pojedynczej aplikacji klienta lub pojedynczej maszyny klienckiej. Podawanie PIDu i hostname'a umożliwia np. administratorowi zatrzymanie konkretnego procesu, który obciąża zbytnio serwer.

Oczywiście — jak zawsze, ale tu szczególnie — w żaden sposób nie należy informacjom podawanym tutaj ufać. Zatem np. decyzję o 'dostępnych' metodach logowania serwer powinien podejmować na podstawie adresu IP pobranego z danych połączenia — a nie na podstawie hostname'a przesłanego w tym pakiecie.

Cechy regionalne: Strefa czasowa Strefa czasowa będzie domyślną strefą w której serwer będzie podawał i przyjmował godziny i daty (gdy są one pozbawione informacji o strefie, której dotyczą).

W tym przypadku ponownie (z konieczności — (patrz: B.3.6)) posługujemy się strefą czasową zapisaną w postaci różnicy czasu między czasem lokalnym, a czasem GMT.

Cechy regionalne: Porównywanie napisów (collation) Pole „collation" służy przekazaniu informacji o obowiązującej metodzie porównywania napisów. W obecnej wersji systemu LoXiM pole to nie będzie wykorzystywane, ale już zapewniamy na jago potrzeby 64bity (np. pierwsze 32 bity mogą posłużyć do określenia języka, który stosujemy, a drugie 32 bity mogą być wykorzystywane na flagi (np. czy $A > a$, czy może $A = a$)). W szczególności przekazując te dane (język i zestaw flag) do algorytmu „Unicode Technical Standard #10, Unicode Collation Algorithm" [UTS10], można otrzymać zasady porównywania napisów odpowiednie dla wybranego języka.

Z algorytmu UTS wynika, że drugie 32 bity można wykorzystać w następujący sposób do jego parametryzowania (podajemy numery najmniej znaczących bitów od 0 do 31:

- **0-3** — Te 4 bity wykorzystujemy do przechowania własności „Level", czyli liczby cech uwzględnianych przy sortowaniu. UTS przewiduje następujące 5 poziomów:
  - **0** — zasada domyślna dla języka
  - **1** — L1 (Base letters)
  - **2** — L2 (L1+Accents)
  - **3** — L3 (L2+Case)
  - **4** — L4 (L3+Punct)
  - **5** — L5 (L4+Codepoint)
- **4-5** — Te dwa bity wykorzystamy do przechowania zachowania względem porównywania znaków wielkich i małych. Proponujemy by:
  - **0** — Nie wymuszaj (domyślnie dla języka)
  - **1** — Uznawaj wielkie i małe litery za tożsame

2 — Wielkie litery pierwsze

3 — Małe litery pierwsze

6 Jeśli bit zapalony, to używamy opcji „French accents”

7 Jeśli bit zapalony, to używamy opcji „Add case Level”

8 Jeśli bit zapalony, to używamy opcji „Full normalization mode”

9 Jeśli bit zapalony, to używamy opcji „Add Hiragana Level”

10 Jeśli bit zapalony, to używamy opcji „Numeric Collation”

11-31 aktualnie nie używane

Cechy regionalne: Język klienta Język klienta służy tylko umożliwieniu przesyłania komunikatów (takich jak komunikaty o błędach) w języku możliwie bliskim językowi użytkownika. Jeśli język wybrany tą opcją nie jest wspierany, to system wybiera inny — możliwie bliski wybranemu lub domyślny (np. angielski).

Kodowanie znaków nie jest przesyłane — ze względu na założenie, że wszelka komunikacja tekstowa będzie prowadzona za pomocą kodowania UTF-8 (Big-endian). Zatem zawsze do klienta należy przekodowanie takiego napisu z i na stronę kodową klienta.

Głębszych rozważań na poziomie implementacji serwera bazy danych wymaga kwestia traktowania danych regionalnych takich jak strefa czasowa, czy metoda porównywania napisów.

Niektóre serwery baz danych dokonują w różny sposób konwersji danych zawartych w bazie danych na czas lokalny “sesji” użytkownika. Protokół zaleca, by takie zachowanie podlegało konfiguracji za pomocą opcji obsługiwanych przez pakiet S-C-SETOPT (patrz: B.5.8).

Podobną kwestią jest to, na jakim poziomie baza danych powinna pamiętać metodę porównywania napisów. Relacyjne bazy danych takie jak MySQL i MSSQL pamiętają ją na poziomie kolumny w tabeli), co jest dość elastycznym rozwiązaniem (choć czasem zaskakuje programistę, gdy odkrywa, że przeszukiwania po jednej kolumnie są czułe na wielkość znaków, a po drugiej kolumnie w tej samej tabeli nie są). Występującym, ale uzasadnionym problemem w tym rozwiązaniu jest to, że nie można porównywać wartości w dwóch kolumnach o różnych metodach porównywaniu napisów (w tej sytuacji wzorowa baza danych powinna udostępnić mechanizm specyfikacji według której kolacji to porównanie ma działać — z czym w praktyce się jeszcze nie spotkałem).

Brak schematu w modelu AS_0 bazy Loxim uniemożliwia takie zapamiętywanie informacji o „collation”. Zapamiętywanie tej informacji z każdym napisem wydaje się być skrajnie niepraktyczne (duży koszt pamięciowy i niska potencjalna użyteczność). Za podporządkowaniem metody porównywania napisów do ustawień sesji przemawia sytuacja, gdy użytkownik przyzwyczajony do języka np. szwedzkiego prosi bazę danych o przygotowaniu mu posortowanej listy jego międzynarodowych kontrahentów i oczekuje, że dostanie tę bazę posortowaną według wytycznych jego języka. Z analogicznym roszczeniem do tej bazy może wystąpić Niemiec — dla którego naturalne zasady sortowania są nieco inne. Dlatego w zależności od ustawienia aplikacji klienckiej (a zatem sesji) powinna być wybrana metoda porównywania napisów.

Z drugiej strony — nie jest rzeczą właściwą, gdy zapytanie (Emp where title='Programmer') da nam różne wyniki w zależności od ustawień naszej aplikacji klienckiej (np. w jednym przypadku z uwzględnieniem wielkości znaków, a w drugim bez uwzględnienia wielkości znaków). Pewnym rozwiązaniem w tym przypadku jest specyfikowanie metody na poziomie np. korzeni (rootów).

Protokół nie specyfikuje, które rozwiązanie jest słuszne, ale udostępnia pole, które może być wykorzystane do przesłania tych danych. Także przy zastosowaniu zarówno jednego jak

i drugiego rozwiązania pole to może być źródłem danych dla odpowiedniej konfiguracji nowo tworzonych obiektów.

Format pakietu W-C-Hello jest następujący:

Od - do	**Typ/Wartość**	**Zawartość**
-5 → -5	10 (uint8)	Identyfikator paczki
-4 → -1	a (uint32)	Długość właściwej zawartości paczki
0 → 7	sint64	PID procesu klienta — może być 0, gdy nie obsługiwany
8 → k	sstring	Nazwa programu klienckiego.
$k+1 \rightarrow l$	sstring	Wersja programu klienckiego.
$l+1 \rightarrow m$	sstring	Hostname — nazwa komputera (z domeną).
$m+1 \rightarrow m+3$	sstring (3 znaki)	Język użytkownika — kod literowy według standardu ISO-639-2 (http://www.loc.gov/standards/iso639-2/php/code_list.php)
$m+4 \rightarrow m+11$	uint64	Kolacji
$m+12 \rightarrow m+12$	sint8	Strefa czasowa klienta w postaci liczby całkowitej opisującej przesunięcie względem GMT. Czyli dopuszczalne wartości to od -14 (to nie jest błąd) do +12.

W-S-HELLO

Paczka służy przedstawieniu możliwości protokołu i podstawowych informacji o serwerze.

Zatem jego format jest następujący:

Od - do	Typ/Wartość	Zawartość
-5 → -5	11 (uint8)	Identyfikator paczki
-4 → -1	*a* (uint32)	Długość właściwej zawartości paczki
0 → 0	p_major=2(uint8)	Numer główny (major) wersji protokołu
1 → 1	p_minor=0(uint8)	Numer pomocniczy (minor) wersji protokołu
2 → 2	s_major (uint8)	Numer główny (major) wersji systemu
3 → 3	s_minor (uint8)	Numer pomocniczy (minor) wersji systemu
4 → 7	max_package_size (uint32)	Rozmiar maksymalnego paczki przesyłanego tym protokołem — powinno być > 1024 i wartość powyżej 1048586 powinna być istotnie uzasadniona (wartość jest odczytywana z konfiguracji serwera)
8 → 15	features (uint64)	Mapa bitowa dostępnych cech serwer'a
16 → 23	auth_methods (uint64)	Mapa bitowa dostępnych metod autoryzacji
24 → 43	salt (char[20])	160 bitowy ciąg losowy — używany przez niektóre metody autoryzacji

Dostępne features dzielą się na:

Tryby transmisji i działania protokołu: 0x0001=F_SSL — połączenie może być szyfrowane metodą SSL

0x0002=F_O_SSL — obligatoryjne połączenie szyfrowane metodą SSL (wymusza też obecność flagi F_SSL)

0x0004=F_ZLIB — połączenie może być kompresowane za pomocą biblioteki ZLIB

Tryb przetwarzania zapytania: 0x0010=F_AUTOCOMMIT — serwer udostępnia tryb autocommit (każde zapytanie zadane poza transakcją rozpoczyna swoją transakcję, które jest automatycznie zamykana po wykonaniu polecenia)

0x0020=F_OPTIMALIZATION — można włączyć optymalizator zapytań

. . . np. poziomy izolacji transakcji

Aktualnie przewidziane metody autentykacji to auth_methods to: (patrz: B.7.6)

0x0001=AM_TRUST — serwer uwierzy w każdą podaną tożsamość ((patrz: B.7.6))

0x0002=AM_MYSQL5_AUTH — autoryzacja metodą stosowaną przez serwer ((patrz: B.7.6)) MySQL5 [MYSQL] — w oparciu o przesyłanie i przechowywanie skrótu hasła algorytmem SHA1. Generalnie schemat jest następujący:

Serwer przechowuje: $SHA1(password)$

Klient przesyła: $SHA1(password) XOR (SHA1(salt.SHA1(SHA1(password))))$

W-C-MODE

Jest to paczka służąca do przejścia w protokole wstępnym na inny tryb transmisji. Obecnie obejmuje to koncepcję szyfrowania, kompresji, a także ewentualnie transmisji w formacie XML.

Jedynym parametrem paczki jest wybrany tryb transmisji. Zatem format paczki jest następujący:

Od - do	Typ/Wartość	Zawartość
-5 → -5	12 (uint8)	Identyfikator tej paczki
-4 → -1	8 (uint32)	Długość właściwej zawartości paczki
0 → 7	nowy_tryb (uint64)	JEDNA ze stałych trybu transmisji:

Przewidywane tryby transmisji to:

TT_SSL=1 komunikacja szyfrowana z wykorzystaniem protokołu SSL

TT_ZLIB=2 komunikacja kompresowana

W kolejne tryby należy wchodzić pojedynczo. Nowy tryb umieszczany jest na szczycie stosu (warstwa transportu znajduje się na spodzie stosu). Zatem, aby sensownie uruchomić jednocześnie kompresję i szyfrować, należy najpierw uruchomić szyfrowanie, a później kompresję. W ten sposób dane zostaną najpierw skompresowane, a później zaszyfrowane.

Dostępne odpowiedzi to:

S-SC-OK Polecenie zostało zaakceptowane. Za chwile nastąpią negocjacje w kwestii ustalenia nowego trybu. Szczegóły ustalenia konkretnego trybu zależą od wybranego trybu. Strona, która inicjalizuje dalsza konwersację także zależy od konkretnego elementu (np. w przypadku SSL'u serwer powinien rozpocząć „HANDSHAKE").

Jeśli proces negocjacji nowego trybu zakończy się porażką, to połączenie musi zostać natychmiast zerwane.

S-SC-ERROR Nie jest możliwe przejście to żądanego trybu z jakiegoś powodu. Spodziewane komunikaty odpowiedzi to może być:

ERR-ModeNotAvoilable ModeNotAvoilable.

ERR-ModeAlredySet ModeAlredySet.

ERR-Internal Internal.

W-C-LOGIN

Komunikat służy przekazaniu przez klienta serwerowi informacji o tym, za pomocą którego mechanizmu logowania klient będzie się chciał zalogować do serwera. W praktyce jedynym parametrem paczki jest identyfikator wybranej metody autoryzacji B.7.6). Zatem format paczki jest następujący:

Od - do	Typ/Wartość	Zawartość
-5 → -5	13 (uint8)	Identyfikator paczki login
-4 → -1	8 (uint32)	Długość właściwej zawartości paczki
0 → 7	nowy_tryb (uint64)	JEDNA z wartości metody autoryzacji (patrz: B.7.6)

W momencie wysłania tej paczki — zarządzanie protokołem jest oddane procesowi realizującemu konkretną metodę autoryzacji. Metoda taka powinna zapewnić to, że serwer po jej realizacji wyśle pakiet W-S-AUTHORIZED lub zgłosi błąd i zakończy połączenia, a klient po zakończeniu działanie tej metody, będzie wiedział, że ona się zakończyła i będzie gotowy na odbiór pakietu W-S-AUTHORIZED lub informacji o błędzie.

W sytuacji, gdy serwer otrzyma paczkę W-C-LOGIN chcącą przeprowadzić autoryzację metodą nie wspieraną przez serwer — serwer powinien natychmiast zerwać połączenie.

W-S-AUTHORIZED

Jest to paczka potwierdzająca skuteczną autoryzacje i oświadczający o zakończeniu pracy serwera w trybie wstępnym i o mówiący przejściu w tryb pracy właściwej.

Format jest następujący:

Od - do	Typ/Wartość	Zawartość
-5 → -5	14 (uint8)	Identyfikator tej paczki
-4 → -1	0 (uint32)	Długość właściwej zawartości paczki

W-C-PASSWORD

Jest to paczka w którym klient autoryzujący się przy pomocy hasła powinien przesłać swój login i hasło. W szczególności ta paczka wykorzystuje metody autoryzacji: Trust oraz MySQLpassword (patrz: B.7.6).

Format jego jest następujący:

Od - do	Typ/Wartość	Zawartość
$-5 \rightarrow -5$	15 (uint8)	Identyfikator tej paczki
$-4 \rightarrow -1$	m (uint32)	Długość właściwej zawartości tego paczki
$0 \rightarrow n$	login (sstring)	Login autoryzującego się użytkownika
$n+1 \rightarrow m-1$	password (bytes)	Hasło lub jego skrót lub NULL w przypadku metody Trust

B.5.5. Protokół właściwy — obsługa zapytania

Q-C-STATEMENT

Paczka służy do przesłania zapytania na serwer. Jeżeli flaga EXECUTE jest zapalona — to znaczy, że przesyłamy proste — bezparametrowe zapytanie — które chcemy by zostało

natychmiast wykonane przez serwer. W przeciwnym przypadku przesyłamy tylko zapytanie — by zostało sparsowany — i by został my nadany numer StetementId. Posługując się później tym numerem będziemy mogli wielokrotnie zbindować parametry do zapytania i je wykonać (Q-C-EXECUTE (patrz: B.5.5)).

Format paczki jest następujący:

Od - do	**Typ/Wartość**	**Zawartość**
$-5 \rightarrow -5$	64 (uint8)	Identyfikator tej paczki
$-4 \rightarrow -1$	m (uint32)	Długość właściwej zawartości tej paczki
$0 \rightarrow 7$	flags (uint64)	Flagi ustawiające opcje zapytania (patrz: dostępne flagi poniżej)
$8 \rightarrow m-1$	statement (string)	Zapytanie wysyłane na serwer

Aktualnie przewidziane flagi to:

EXECUTE = 0x0001 — jeśli flaga jest zapalona to zapytanie zachowuje się tak, jakby natychmiast po nim została wysłana Q-C-EXECUTE (patrz: B.5.5). Czyli zamiast odpowiedzi Q-S-STMTPARSED (patrz: B.5.5) pojawi się albo odpowiedź Q-S-EXECUTING (patrz: B.5.5), albo zostanie zgłoszony któryś z błędów charakterystycznych dla poleceń Q-C-STATEMENT (patrz: B.5.5) i Q-C-EXECUTE (patrz: B.5.5).

READONLY = 0x0002 — jeśli flaga jest zapalona to zapytanie nie ma prawa wprowadzać żadnych modyfikacji w bazie danych

W odpowiedzi na tę paczkę może przyjść paczka Q-S-STMTPARSED (patrz: B.5.5) (jeśli nie była podniesiona flaga EXECUTE) albo (Q-S-EXECUTING (patrz: B.5.5) — jeśli była podniesiona flaga EXECUTE) albo jeden z poniższych błędów:

ERR-SyntaxError SyntaxError.

ERR-OperationNotAllowed OperationNotAllowed.

ERR-Internal Internal.

oraz błędy charakterystyczne dla Q-C-EXECUTE (patrz: B.5.5) — o ile flaga EXECUTE jest zapalona

Q-S-STMTPARSED

Paczka informuje, że analiza składniowa zapytania się powiodła i, że zapytaniu został nadany StatementId. Paczka jest wysyłana w sposób synchroniczny w odpowiedzi na komunikat Q-C-STATEMENT (patrz: B.5.5).

Format jest następujący:

Od - do	Typ/Wartość	Zawartość
$-5 \rightarrow -5$	65 (uint8)	Identyfikator tej paczki
$-4 \rightarrow -1$	16 (uint32)	Długość właściwej zawartości tej paczki
$0 \rightarrow 7$	statementId (uint64)	Id nadane temu zapytaniu/poleceniu
$8 \rightarrow 15$	paramsCnt (uint32)	Liczba parametrów do ustalenia w sparsowanym zapytaniu.

Q-C-EXECUTE

Paczkę tę wysyła klient w celu poinformowania serwera o tym, że chce wykonać zadane poprzez statementId zapytanie. Często, także tym zapytaniem będzie się dokonywało bindowania bindowania poszczególnych parametrów zapytania z identyfikatorami wartości w schowku (patrz obsługa schowka) w celu wykonania zapytania.

Format paczki jest następujący:

Od - do	Typ/Wartość	Zawartość
$-5 \rightarrow -5$	66 (uint8)	Identyfikator tej paczki
$-4 \rightarrow -1$	m (uint32)	Długość właściwej zawartości tej paczki
$0 \rightarrow 7$	statementId (uint64)	Id nadane temu zapytaniu/poleceniu
$8 \rightarrow 15$	flags (uint32)	Flagi — wskazówki dotyczące wyników zapytania — patrz niżej
$16 \rightarrow 19$	paramsCnt (uint32)	Liczba parametrów do ustalenia w sparsowanym zapytaniu.
$m_{i-1} \rightarrow m_i - 1$	$valueId_i$ dla $i \in (1..paramsCnt)$ (varuint)	Id będące i-tym parametrem. Id powinno się odnosić do Id wartości znajdującym się w schowku sesji — ustalonym uprzednio poprzez paczkę V-SC-SENDVALUE (patrz: B.5.6)

Przewidywane flagi to (ciąg dalszy do flag z Q-C-STATEMENT):

0x0100 PREFER-DFS Oznacza, że użytkownik sugeruje by wyniki były przesyłane w kolejności umożliwiającej szybką konstrukcję odpowiedzi metodą DFS. Wyklucza się z PREFER-BFS.

0x0100 PREFER-BSF Oznacza, że użytkownik sugeruje by wyniki były przesyłane w kolejności umożliwiającej szybką konstrukcję odpowiedzi metodą BFS. Wyklucza się z PREFER-DFS.

Paczka w sposób synchroniczny jest związana z odpowiedzią Q-S-EXECUTING (patrz: B.5.5) lub jednym z następujących błędów:

ERR-ParamsIncomplete ParamsIncomplete.

ERR-NoSuchValueId NoSuchValueId.

ERR-OperationNotPermited OperationNotPermited.

ERR-Internal Internal.

Q-S-EXECUTING

Paczka ta jest synchronicznym powiadomieniem o przyjęciu do przetworzenia danego zapytanie (czyli jest wysyłana w odpowiedzi na pakiety Q-C-EXECUTE (patrz: B.5.5) oraz Q-C-STATEMENT (patrz: B.5.5) (z flagą EXECUTE)).

Format pakietu jest następujący:

Od - do	Typ/Wartość	Zawartość
-5 → -5	67 (uint8)	Identyfikator paczki EXECUTING
-4 → -1	0 (uint32)	Długość właściwej zawartości paczki

Otrzymanie tej paczki świadczy o tym, że klient znalazł się w trybie wykonywania zapytania (tzn. że nie możemy wysłać nowego zapytania do czasu przetworzenia lub anulowania bieżącego zapytania, a także to, że aktualnie otrzymywane pakiety z grupy V-. . . dotyczące wyników zwracanych przez to zapytanie).

Q-S-EXECUTION-FINISHED

Paczka informuje, że bieżące polecenie się zakończyło i ewentualnie informuje o liczbie zmian dokonanych przez to zadanie.

Format pakietu jest następujący:

Od - do	Typ/Wartość	Zawartość
$-5 \rightarrow -5$	70 (uint8)	Identyfikator paczki operationOk
$-4 \rightarrow -1$	a (uint32)	Długość właściwej zawartości paczki
$0 \rightarrow d-1$	modAtomPointerCnt (varuint)	liczba zmodyfikowanych obiektów atomowych i pointerowych przez to zdanie (jeśli nie jest określona to serwer ma to zwrócić NULL)
$d \rightarrow b-1$	delCnt (varuint)	liczba usuniętych obiektów (jeśli nie jest określona to serwer ma to zwrócić NULL)
$b-1 \rightarrow c$	newRootsCnt (varuint)	liczba utworzonych obiektów korzeniowych (jeśli nie jest określona to serwer ma to zwrócić NULL)
$c-1 \rightarrow a-1$	insertsCnt (varuint)	liczba obiektów wstawionych do obiektów złożonych (również tych nowo utworzonych) (jeśli nie jest określona to serwer ma to zwrócić NULL)

Wysłanie (ale nie umieszczenie w kolejce paczek do wysłania) tej paczki przez serwer powoduje, że wychodzi on z trybu przetwarzania zapytania.

Wysyłanie ilości dokonanych zmian ma ułatwić pracę systemów korzystających z mechanizmu tzw. optymistycznego przetwarzania transakcji przez aplikację kliencką (serwery aplikacyjne i mechanizmy w stylu „hibernate").

B.5.6. Protokół właściwy — przesyłanie wartości

V-SC-SENDVALUES

Paczka informuje drugą stronę, że rozpoczyna się transfer (zbioru) wartości. Może zawierać informacje o orientacyjnej ilości przesyłanych wyników — by umożliwić wyświetlenie użytkownikowi orientacyjnej informacji o postępie.

Format paczki jest następujący:

Od - do	**Typ/Wartość**	**Zawartość**
$-5 \rightarrow -5$	32 (uint8)	Identyfikator paczki operationOk
$-4 \rightarrow -1$	a (uint32)	Długość właściwej zawartości paczki
$0 \rightarrow l$	rootValueId (varuint)	ID korzenia — paczki, która będzie zawierała właściwy obiekt z wartością. Wartość NULL nie jest dozwolona.
$l+1 \rightarrow m$	oBundlesCount (varuint)	Orientacyjne liczba paczek, które zostaną przesłane — NULL — jeśli nie znana
$m+1 \rightarrow n$	oBidCount (varuint)	Orientacyjna liczba obiektów, które zostaną przesłane — NULL — jeśli jeszcze nie określona
$n+1 \rightarrow a-1$	pVidCount (varuint)	Dokładna liczba obiektów, które zostaną przesłane — NULL — jeśli jeszcze nie określona

V-SC-SENDVALUE

Paczka ta służy do przesłania pewnej wartości. Każda wartość posiada swoje ID — generowane przez stronę wysyłającą. ID to będzie służyło do budowania bardziej złożonych wartości poprzez grupowanie w odpowiednich strukturach ID-ków identyfikujący elementy leżące głębiej w hierarchii. Nie ma żadnego wymogu, aby ID do których się odwołujemy były przesłane wcześniej niż obiekt z nich korzystający.

Przez wartość rozumiemy każdą konstrukcję zgodną z konstrukcji wartości podaną w [SBQL] w rozdziale „SBA: Environment Stack in the AS_0 Store Model — Result returned by Queries".

Format tych danych jest następujący:

Od - do	Typ/Wartość	Zawartość
$-5 \rightarrow -5$	33 (uint8)	Identyfikator paczki operationOk
$-4 \rightarrow -1$	a (uint32)	Długość właściwej zawartości paczki
$0 \rightarrow l$	ValueId (varuint)	Identyfikator wartości
$l+1 \rightarrow l+1$	Flags (uint8)	Flagi z informacjami o pakiecie
$l+2 \rightarrow m$	Typ wartości	Jedna ze stałych kodowych — opisanych poniżej
$m+1 \rightarrow a-1$	data (zależne od typu)	Dane właściwe wartości — zależne od typu wskazanego przez poprzednią komórkę

Przewidywane flagi to:

TO-BE-CONTINUED=0x01 Następny pakiet będzie zawierał ciąg dalszy informacji do tego pakietu (w tym pakiecie nie zmieściły się wszystkie dane, które miały być przesłane — np. ze względu na ograniczenie max_package_size). Następny pakiet będzie dotyczył tego samego obiektu (ten sam ValueId), ale będzie zawierał dalsze wartości.

Przewidywane typy proste to (nie wszystkie typy muszą być dostępne dla użytkowników):

Nazwa	Kod	Opis	Rozdzielalny
UINT8	0x0001	Zapisany jako uint8	-
SINT8	0x0002	Zapisany jako sint8	-
UINT16	0x0003	...	-
SINT16	0x0004	...	-
UINT32	0x0005	...	-
SINT32	0x0006	...	-
UINT64	0x0007	...	-
SINT64	0x0008	...	-
BOOL	0x0009	Zapisany jako bool	-
DATE	0x000A	Data	-
TIME	0x000B	Czas bez strefy czasowej	-
DATETIME	0x000C	Data i czas bez strefy czasowej	-
TIMETZ	0x000D	Czas ze strefą czasową	-
DATETIMETZ	0x000E	Data i czas ze strefą czasową.	-
BYTES	0x000F	Obiekt binarny (nie musi być długi)	+
VARCHAR	0x0010	Tekst (zakodowany w UTF-8 Big-endian).	+
DOUBLE	0x0011	Dane rzeczywiste podwójnej precyzji	-

Przewidywane typy organizujące strukturę (Zgodnie z [SBQL] „SBA: Enviroment Stack in the AS_0 Store Model — Results returned by queries”):

Nazwa	Kod	Opis	Rozdzielalny
VOID	0x0080	Typ pusty.	-
LINK	0x0081	Taka wartość jak opisana poprzez pakiet o danym valueId	-
BINDING	0x0082	Związanie poprzez daną nazwę danej wartości	-
STRUCT	0x0083	Struktura elementów (nie stanowiąca kolekcji)	+
BAG	0x0084	Kolekcja będąca multizbiorem elementów	+
SEQUENCE	0x0085	Kolekcja będąca ciągiem elementów	+
REF	0x0086	Odwołanie do pewnego miejsca w strukturze danych (najprawdopodobniej poza przesłanym pakietem), raczej do krótkotrwałych operacji	-
EXTERNAL_REF	0x0087	Odwołanie do pewnego miejsca w strukturze danych (najprawdopodobniej poza przesłanym pakietem), w celu w przechowywania w innym zewnętrznym systemie	-

W celu poznania struktury poszczególnych formatów danych, należy zapoznać się z rozdziałem B.6.

V-SC-FINISHED

Paczka informuje stronę odbierającą, że wszystkie wartości, które miały zostać przesłane — zostały już wysłane. Oczekuje, że strona odbierająca sprawdzi spójność danych i odpowie synchronicznie, bądź poprzez S-SC-OK (patrz: B.5.8) bądź poprzez zgłoszenie błędu. W obu sytuacjach jednak wysłanie tej paczki powoduje zakończenie tryby przesyłania wyniku.

Format paczki jest następujący:

Od - do	Typ/Wartość	Zawartość
-5 → -5	34 (uint8)	Identyfikator paczki
-4 → -1	0 (uint32)	Długość właściwej zawartości paczki

V-SC-ABORT

Paczka może być wysłana bądź przez stronę odbierającą, bądź przez wysyłającą. Oznacza ona, że należy zaprzestać wysyłać lub oczekiwać na dane.

Paczka ta może być także wysłana poza trybem przesyłania wartości przez serwer - i informuje wtedy o przerwaniu wykonywania (obsługi) bieżącego zapytania.

Wysłanie jej w trybie wykonywania zapytania powoduje opuszczenie tego trybu.

Format paczki jest następujący:

Od - do	Typ/Wartość	Zawartość
$-5 \rightarrow -5$	35 (uint8)	Identyfikator paczki
$-4 \rightarrow -1$	n (uint32)	Długość właściwej zawartości paczki
$0 \rightarrow 3$	reasonCode (uint32)	Kod błędu mówiący o przyczynie przerwania zapytania, 0 — brak uzasadnienia
$4 \rightarrow n-1$	reasonString (sstring)	Słowny opis powodu (najlepiej w języku określonym w fazie wstępnej)

Przewidywane obecnie reasonCody to:

ADMINISTRATION-REASON Twoje zapytanie zostało przerwane na skutek działań administracyjnych

YOU-ARE-TRANSACTION-VICTIM Twoja transakcja została wycofana ze względu na powstanie zastoju (DEADLOCK)

OPERATION-NOT-PERMITED Próba wykonania niedozwolonej operacji

TIME-LIMIT-EXCEEDED Przekroczony maksymalny czas wykonywania zapytania

OUT-OF-MEMORY Zabrakło pamięci na dalsze przetwarzanie wyniku

TYPE-CHECK-ERROR Problem konwersji (dynamicznej kontroli typów)

OTHER-RUN-TIME-ERROR Inny błąd wykonania

B.5.7. Protokół właściwy — paczki różne

A-SC-PING

Paczka jest wysyłany okresowo przez serwer do klienta (lub potencjalnie przez klienta do serwera) w celu stwierdzenia, czy łączność z klientem jest zachowana, a klient się nie zawiesił. Paczka składa się w praktyce tylko z nagłówka.

Zatem jego format jest następujący:

Od - do	Typ/Wartość	Zawartość
-5 → -5	128 (uint8)	Identyfikator paczki ping
-4 → -1	0 (uint32)	Długość właściwej zawartości paczki

Paczka A-SC-PING nie jest obsługiwany w fazie wstępnej. Tam połączenie zostanie automatycznie zamknięte przez serwer po upływie login-timeout.

A-SC-PONG

Paczka wysyłany w odpowiedzi na paczkę A-SC-PING (patrz: B.5.7).

Paczka składa się w praktyce tylko z nagłówka.

Zatem jego format jest następujący:

Od - do	Typ/Wartość	Zawartość
-5 → -5	129 (uint8)	Identyfikator paczki pong
-1 → -4	0 (uint32)	Długość właściwej zawartości paczki

Paczka A-SC-PONG (patrz: B.5.7) nie jest obsługiwany w fazie wstępnej. Tam połączenie zostanie automatycznie zamknięte przez serwer po upływie login-timeout.

B.5.8. Komunikaty ogólne

A-SC-OK

Paczka potwierdzający przyjęcie i zrozumienie komendy wysłanej na serwer przez klienta. Paczka składa się w praktyce tylko z nagłówka.

Zatem jego format jest następujący:

Od - do	Typ/Wartość	Zawartość
-5 → -5	01 (uint8)	Identyfikator paczki ok
-4 → -1	0 (uint32)	Długość właściwej zawartości paczki

A-SC-ERROR

Format paczki jest następujący:

Od - do	Typ/Wartość	Zawartość
-5 → -5	02 (uint8)	Identyfikator paczki error
-4 → -1	$12 + n$ (uint32)	Długość właściwej zawartości paczki
0 → 3	uint32	Kod błędu
4 → a	varuint	Numer jednostki wykonywania (np. StatementId lub Portalu lub NULL) — z którą związany jest błąd
$a + 1 \rightarrow n$	sstring	Opis błędu — słowny
$n + 1 \rightarrow n + 4$	uint32	Numer linii w której błąd wystąpił (0 — gdy nie dotyczy)
$n + 5 \rightarrow n + 8$	uint32	Numer znaku w linii w której błąd wystąpił (0 — gdy nie dotyczy)

A-SC-BYE

Paczka zostaje wysyłana, by poinformować drugą stronę o zakończeniu połączenia. Powinna zostać wysłana, gdy w tym protokole jest napisane, że strona może zakończyć połączenie.

Strona, która otrzymuję tę paczkę powinna zamknąć połączenie po jej odczytaniu (bez wysyłania zwrotnego A-SC-Bye).

Jeśli jest napisane, że strona może/musi zerwać połączenie to ta paczka nie może zostać wysłana, tylko połączenie musi zostać zamknięte na poziomie TCP/IP.

Format paczki jest następujący:

Od - do	Typ/Wartość	Zawartość
-5 → -5	03 (uint8)	Identyfikator paczki bye
-4 → -1	0 (uint32)	Długość właściwej zawartości paczki
0 → n	reason (string)	Powód zakończenia połączenia — do celów informacyjnych. Może być NULL.

S-C-SETOPT

W protokole wstępnym ta paczka służy do ustawienia opcji przez klienta, które mogą być istotne w procesie stwierdzania, czy klient ma dostęp dla danego zasobu, czy go niema. .

W praktyce polecenie to przesyła parę dwóch stringów: klucz i wartość. Zatem konstrukcja paczki jest następująca:

Od - do	**Typ/Wartość**	**Zawartość**
$-5 \rightarrow -5$	130 (uint8)	Identyfikator paczki setopt
$-4 \rightarrow -1$	8 (uint32)	Długość właściwej zawartości paczki
$0 \rightarrow n$	key (sstring)	Klucz
$n+1 \rightarrow m$	value (string)	Wartość nadana kluczowi

Z punktu widzenia LoXiM'a sensownym się wydaje wprowadzenie w protokole wstępnym następujących parametrów:

local_root Wskazuje, który obiekt stanowi root'a dla tego połączenia, a zatem wszystkie łączenia (bindings) znajdujące się w środowisku początkowej ewaluacji SBQL'a będą pochodziły z niego. Coś w stylu Unikowego polecenia 'chroot'.

W protokole właściwym natomiast:

autocommit Dopuszczalne wartości to 'true' i 'false'. Ustawia opcję autocommit.

Dygresja: Wydaje się sensownym zastąpienie w protokole właściwym tego polecenia poprzez czystego SBQL'a odwołującego się do wirtualnych wpisów w bazie danych.

Można sobie wyobrazić następujące zapytanie SBQL: $$.session.autocommit:=true, gdzie $$ jest wirtualnym korzeniem (coś na podobieństwo Unix'owego katalogu /proc)

B.6. Złożone typy danych

B.6.1. VOID

Dane tego typu paczki nie istnieją (ich długość wynosi 0).

B.6.2. LINK

Zawiera informację, że wartość ta jest opisana w innej paczce o zadanym ID.

Od - do	**Typ/Wartość**	**Zawartość**
$0 \rightarrow$ a	*valueId* (varuint)	Identyfikator paczki w której jest właściwa wartość

Ten typ będzie używany przeważnie w innych (poniższych) złożonych strukturach w celu uniknięcia rekurencyjnego budowania „dużych pakietów".

B.6.3. BINDING

Jest to związanie pewnej wartości poprzez konkretną nazwę wewnątrz obiektu.

Od - do	**Typ/Wartość**	**Zawartość**
$1 \rightarrow a$	*bindingName* (sstring)	Nazwa z którą wiążemy daną wartość
$a + 1 \rightarrow b$	*type* (varuint)	Typ wartości do którego jest binding
$b + 1 \rightarrow c$	(format zależy od *type*)	Stosowna wartość (format zależny od typu)

FEATURE: W następnych wersjach można rozważyć kompresje powtarzających się bindingów (by nie przechowywać ich jako "string" ale jako słownik bindingów i przesyłać tylko id binding'a)

Zatem propozycja alternatywnego formatu przesyłania bindingów jest taka:

Od - do	**Typ/Wartość**	**Zawartość**
$0 \rightarrow 0$	NULL (sstring)	
$1 \rightarrow a$	*valueId* (varuint)	Identyfikator wcześniej wysłanego (w tej grupie wartości) bindingu
$a + 1 \rightarrow b$	*type* (varuint)	Typ wartości do którego jest binding
$b + 1 \rightarrow c$	(format zależy od *type*)	Stosowna wartość (format zależny od typu)

Oba te formaty są ze sobą zgodne. Jeśli dany string ma wartość NULL, to znaczy, że mamy do czynienia z drugim formatem.

B.6.4. STRUCT, BAG, SEQUENCE

Wszystkie te trzy formaty danych "póki co" mają ten sam format.

Przewidujemy dwie możliwości zapisywania takich struktur — jednorodną i różnorodną. Celem wprowadzenia formatu jednorodnego jest uniknięcie przesyłania przy każdym elemencie zbioru informacji o jego typie — poprzez podanie jednego globalnego typu dla wszystkich elementów. Formę jednorodną wyróżnia nie null'owy „typ globalny".

Zatem format typu jednorodnego to:

Od - do	**Typ/Wartość**	**Zawartość**
$0 \rightarrow a$	count (varuint)	liczba wartości zawartych **w tym pakiecie**
$a+1 \rightarrow b$	*globalType* (varuint)	Typ wszystkich elementów kolekcji
$b+1 \rightarrow c_1$	(zależny od „globalType")	Wartość 1-wsza
$c_1+1 \rightarrow c_2$	(zależny od „globalType")	Wartość 2-ga
$\ldots \rightarrow \ldots$	$\ldots$	$\ldots$
$c_{count}+1 \rightarrow c_{count+1}$	(zależny od „globalType")	Wartość ostatnia

Analogicznie format pakietu różnorodnego uzyskujemy poprzez przesunięcie pole "globalType" do każdej pojedynczej wartości:

Od - do	**Typ/Wartość**	**Zawartość**
$0 \rightarrow a$	count (varuint)	liczba wartości zawartych **w tym pakiecie**
$a+1 \rightarrow b$	*NULL* (varuint)	Typ wszystkich elementów kolekcji
$t_1 \rightarrow c_1 - 1$	$type_1$ (varuint)	Typ wartości 1-wszej
$c_1 \rightarrow t_2 - 1$	(zależny od „$type_1$")	Wartość 1-wsza
$t_2 \rightarrow c_2 - 1$	$type_2$ (varuint)	Typ wartości 2-giej
$c_2 \rightarrow t_3 - 1$	(zależny od „$type_2$")	Wartość 2-ga
$\ldots \rightarrow \ldots$	$\ldots$	$\ldots$
$t_{count}+1 \rightarrow c_{count}-1$	$type_{count}$	Typ ostatniej wartości
$c_{count} \rightarrow c_{count+1}$	(zależny od „$type_{count}$")	Wartość ostatnia

Pole tego typu ten może być rozłożony na wiele pakietów przesyłających wartość — poprzez zaznaczenie flagi TO_BE_CONTINUED i przesłanie identycznego nagłówka wartości dla wszystkich pakietów).

B.6.5. REFERENCE

Jest to typ zawierający pewną referencję do innego miejsca w strukturze danych przechowywanej w bazie danych. Jest on po prostu wewnętrznym identyfikatorem bazy danych i nie powinien być interpretowany przez program kliencki.

Więc format jest następujący:

Od - do	**Typ/Wartość**	**Zawartość**
$0 \rightarrow 7$	*valueId* (uint64)	Wewnętrzny identyfikator miejsca

B.6.6. EXT_REFERENCE

Jest to typ już zarezerwowany, ale jeszcze nie wprowadzony. Służy do, oprócz trzymania wewnętrznego identyfikatora w systemie LoXiM, przechowywanie stempla czasowego — świadczącego o czasie — w którym podany identyfikator był stworzony. Dzięki temu będzie możliwe po stronie systemu bazy danych powtórne wykorzystywanie identyfikatorów.

Zatem wstępnie jego format ustalamy w następujący sposób:

Od - do	**Typ/Wartość**	**Zawartość**
$0 \rightarrow 7$	*valueId* (uint64)	Wewnętrzny identyfikator miejsca
$8 \rightarrow 15$	*stamp* (uint64)	Stempel czasowy lub coś w tym stylu. Ale implementacja klienta i tak nie musi tego interpretować — a nawet nie powinna.

B.7. Bezpieczeństwo

Protokół sieciowy jest z pewnością najbardziej kuszącą częścią systemu — z punktu widzenia osoby chcącej naruszyć jego bezpieczeństwo.

B.7.1. Całkowity brak zaufania

Należy zachować całkowity brak zaufania w kwestiach poprawności przesyłanych danych.

Pod żadnym pozorem, serwer (ani żadna inna aplikacja odczytująca dane) **NIE MOŻE ZAKŁADAĆ, ŻE DOSTARCZANE DANE SĄ POPRAWNE**.

Długości wszystkich buforów i zakresy wartości wszystkich zmiennych myszą być bardzo dokładnie kontrolowane.

Przy stwierdzeniu naruszenie zasad protokołu przez którąś stronę komunikacji, połączenie powinno zostać natychmiast zerwane, a stosowny komunikat powinien zostać zapisany do logów (serwer) lub udostępniony użytkownikowi systemu (klient).

B.7.2. Utrudnienia dla skanerów portów

Zgodnie z opisem protokołu wstępnego system stara się ukryć, to że jest serwerem LoXiM do czasu, aż klient dokona zgodnego z tym protokołem powitania (patrz: B.5.4). Służy to utrudnieniu zdobycia informacji o działających na atakowanym systemie aplikacjach. Dzięki temu skaner portów musi oprócz otworzenia portu, dokonać odpowiedniego zapytania, zanim zostanie poinformowany o opcjach i wersjach protokołu — a zatem o potencjalnych miejscach, o których atakujący może poszukiwać informacji o lukach w zabezpieczeniach.

B.7.3. Synchroniczność/Asynchroniczność

By zmniejszyć dodatkowo pule potencjalnych zagrożeń, ustalamy, że protokół wstępny jest w pełni synchroniczny. Dopiero w protokole właściwym stają się dostępne komunikaty przesyłane przez serwer do klient w sposób asynchroniczny (nie będące odpowiedzią na żądanie klienta).

B.7.4. Limity czasów

Można wprowadzić limit czasów na wykonanie różnych operacji przez użytkownika, których celem utrudnić ataki poprzez nawiązywanie zbyt dużej ilości połączeń, które nawet nie zastają autoryzowane, lub które nazbyt długo pozostają nie używane.

authorization timeout — czas pomiędzy nawiązaniem połączenia, a przeprowadzeniem skutecznego logowania. (opcja może zostać wyłączona).

authorization delay — czas jaki serwer odczeka po nieudanej próbie autoryzacji, zanim poinformuje użytkownika o porażce.

idle time — czas, który jeśli upłynie pomiędzy ostatnią merytoryczną wymianą paczek (i kolejka zadań dla danego połączenia po stronie serwera jest pusta) to serwer może zerwać połączenie. (opcja może zostać wyłączona).

B.7.5. S(B)QL Injection

Jednym z najczęstszych ataków na systemy związane z bazami danych jest atak „SQL injection". Polega on na takim podaniu parametrów dla zapytania, że w wyniku działania oprogramowanie to wygeneruje takie zapytanie, które odniesie inny skutek, niż programista zamierzał. W większości wypadków — poprzez źle wyescapowany tekst - do zapytania dostają się dodatkowe polecenia lub modyfikacje bieżącego zapytania.

Żeby takie działanie utrudnić protokół wprowadza następujące rozwiązania:

1. Tylko jedno polecenie „per paczkę" jest dozwolone. Tzn. że jeżeli w pojedynczym pakiecie zostaną wysłane dwa zapytania (standardowo oddzielone średnikiem) to nie zostanie wykonane z nich żadne).

2. Charakter polecenia: W pakiecie — razem z zapytanemu jest wysyłana informacja, czy jest ono typu 'readonly', czy 'readwrite'. Zapytania readonly nie mają prawa wprowadzić modyfikacji do żadnych obiektów, więc „wstrzyknięte" zapytanie nie dokona nam modyfikacji bazy danych.

Nadal potencjalnie niebezpieczne pozostaną operację zapisujące dane w bazie danych i podatne na sqlinjection, a także będzie istniała możliwość pozyskania dodatkowych danych poprzez wpłynięcie na spectrum wyników zwróconych w zapytaniu.

Należy wspomnieć w tym miejscu także to, że protokół udostępnia mechanizmy parametryzacji zapytań, co nie tylko pozwala uniknąć problemów związanych z wstrzyknięciem złośliwego kodu, ale także pozwala współdzielić plany zapytań przez wiele wywołań zapytania — co pozytywnie wpływa na wydajność.

B.7.6. Metody autoryzacji

Trust (pełne zaufanie)

Brak metody autoryzacji. W praktyce jej dostępność powinna zupełnie wyłączona, a jej istnieje służy tylko umożliwieniu zmiany hasła administratora po tym, jak ten go zapomni (i to tylko dla połączeń z lokalnego interfejsu sieciowego).

Protokół autoryzacji metodą trust wygląda następująco:

1. Klient wysyła paczkę W-C-PASSWORD (patrz: B.5.4) z nazwą użytkownika taką jaką chce otrzymać po zalogowaniu i hasłem ustawionym na NULL.

2. Serwer sprawdza, czy taki użytkownik istnieje i jeśli istnieje, to autoryzacja zostaje potwierdzona poprzez wysłanie do klienta komunikatu W-S-AUTHORIZED (patrz: B.5.4). Serwer przechodzi we właściwy tryb pracy.

   Jeśli natomiast podany użytkownik nie istnieje, to serwer odpowiada błędem ERR-NoSuchUser, po czym zrywa połączenie.

Autoryzacja hasłem — jak w MySQL

Autoryzacja zaszyfrowanym hasłem. Hasło jest także na serwerze przechowywane w postaci zaszyfrowanej.

Protokół autoryzacji metodą MySQLPassword wygląda następująco:

1. Klient pobiera 160 bitów z początku przesłanego przez serwer w komunikacie W-S-HELLO (patrz: B.5.4) zbioru losowych danych 'salt'.

2. Klient pobiera nazwę użytkownika i hasło. Wylicza następującą daną

$$SHA1(password)XOR(SHA1(salt.SHA1(SHA1(password))))$$

.

UWAGA! Zakładamy, że funkcja SHA1 dla danego tekstu (zakodowanego jako UTF-8 Big-endianbez żadnego znaku końca i znacznika długości) zwraca zbiór 160 bitów (czyli 20 bajtów) — a nie tekst z wartością w hex'ach - które jako wynik zwraca wiele funkcji bibliotecznych.

3. Klient wysyła paczkę W-C-LOGIN (patrz: B.5.4) z wybraną procedurą autentykacji.

4. Klient wysyła paczkę W-C-PASSWORD (patrz: B.5.4) z nazwą użytkownika taką jaką chce otrzymać po zalogowaniu i hasłem ustawionym na wyżej wyliczony skrót hasła.

5. Serwer sprawdza, czy taki użytkownik istnieje i jeśli istnieje, to porównuje przesłane hasło z samodzielnie wyliczonym skrótem. to autoryzacja Jeśli wszystko się zgadza — to autoryzacja zostaje potwierdzona poprzez wysłanie do klienta komunikatu W-S-AUTHORIZED (patrz: B.5.4). Serwer przechodzi we właściwy tryb pracy.

   Jeśli natomiast podany użytkownik nie istnieje, to serwer odpowiada błędem ERR-AccessDenied, po czym zrywa połączenie.

B.7.7. Limity

Wydaje się celowym wprowadzenie następujących limitów:

Maksymalna liczba użytkowników (połączeń)

Maksymalny rozmiar paczki przesyłanego w protokole

Maksymalny rozmiar schowka na parametry

B.8. Etapy realizacji projektu

B.8.1. Faza 1

- Zaimplementowanie transportu wszystkich opisanych w tym dokumencie paczek.
- Zaimplementowanie bibliotek umożliwiających nawiązywanie połączeń.

B.8.2. Fazy następne

- Wprowadzenie warstwy kompresji (ZLIB)
- Wprowadzenie warstwy komunikacji szyfrowanej (SSL)
- Przesyłanie informacji o typach na potrzeby (pół-)mocnej kontroli typów.
- Autoryzacja za pomocą mechanizmu Kerberos.
- Autoryzacja za pomocą innych mechanizmów uwierzytelniania.

Dodatek C

Analiza możliwości wykorzystania protokołu LDAP dla SBQL DB

C.1. Wprowadzenie

C.1.1. Cel

Celem tego dokumentu jest przeanalizowanie możliwości wykorzystania protokołu LDAP (Lightweight Directory Access Protocol wersja 3) jako podstawowego narzędzia do komunikacji z semistrukturalną bazą danych z dostępem za pomocą języka SBQL.

W dokumencie zostaną także przedstawione propozycje rozszerzeń protokołu, tak by można było za pomocą protokołu LDAP wraz z opisanymi rozszerzeniami wykorzystać pełną funkcjonalność systemu LoXiM.

C.1.2. Usługi katalogowe

C.1.3. Protokół LDAP a „Usługa katalogowa LDAP" (LDAP service)

Należy rozróżnić dwa pojęcia:

Protokół LDAP — Jest to protokół komunikacyjny — standard specyfikujący zasady komunikacji pomiędzy aplikacją kliencką, a serwerem udostępniającym dane. Wnosi on tylko podstawowe założenia o modelu danych.

Usługa katalogowa LDAP (LDAP (enabled) service) — Jest to usługa z którą można się komunikować przy pomocy protokołu LDAP. Realizuje ona wiele różnych standardów. W szczególności w skład usługi katalogowej LDAP wchodzi obsługa:

- Modelu danych — RFC4512 (Directory Information Models)
- Dostępnych podstawowych metod autoryzacji — RFC4513 (Authentication Methods and Security Mechanism)
- Reprezentacji w postaci napisów nazw znaczących (adresów) - RFC4514 (String Representation of Distinguished Names)
- Reprezentacji zapytań (filtrów wyszukiwania) w postaci napisów - RFC4515 (String Representation of Search Filters)
- Jednolitych wskaźników do zasobów — RFC4516 (Uniform Resource Locator)
- Zasad składniowych i zasad dopasowywania danych — RFC4517 (Syntaxes and Matching Rules)

- Obsługi międzynarodowych napisów — RFC4518 (Internationalized String Preparation)
- Schematu dla aplikacji użytkowych — RFC4519 (Schema for User Applications)

W poniższym dokumencie (o ile nie zaznaczono inaczej) pod terminem LDAP rozumiem „Protokół komunikacyjny LDAP".

C.1.4. Sposoby integracji serwera SBQL z usługą LDAP

Semistrukturalne bazy danych oraz usługi katalogowe łączy wiele cech wspólnych. W przypadku LoXiM'a do najważniejszych z nich należy zaliczyć:

- Obie usługi zajmują się organizacją i udostępnianiem danych według zadanych przez użytkownika kryteriów.
- Obie usługi przechowują dane w strukturze drzewiastej.
- Obie usługi przewidują pojęcie odnośnika pomiędzy węzłami tego drzewa. W przypadku LDAP jest to nazywane „aliasem", a w przypadku struktury SBQL obiektem wskaźnikowym (ang. pointer object).
- Obie usługi przewidują możliwość rozproszenia (przechowywania części drzewa) na różnych serwerach — jednocześnie organizując proces przeszukiwania w ten sposób, że możliwe jest otrzymanie pełnego wyniku dla zapytania dotyczącego większej ilości źródeł danych.

Metody przeszukiwania dla struktury LDAP posiadają następujące cechy

- Wyszukują w danym poddrzewie te WPISY, których wartości atrybutów „pasują" do wskazanych wartości.
- Posiadają bogate mechanizmy definiowania pojęcia „pasowania" (odporność na wielkość znaków, funkcje szyfrujące i porównujące napisy zaszyfrowane).
- Wynikiem zapytania jest zawsze wskazany zbiór atrybutów spośród wszystkich odnalezionych węzłów spełniających wskazany warunek.
- Całkowity brak wsparcia dla przeszukiwań łączących dane pochodzące z różnych węzłów.
- Całkowity brak możliwości uzyskania danych zagregowanych.

W związku z tym, że mechanizm zapytań LDAP jest stosunkowo prosty — umożliwia on bardzo efektywną realizację operacji (w pełni równoległe i z wykorzystaniem niewielkiej pamięci pomocniczej przetwarzanie). Ogromną wadę stanowi natomiast niewielka ilość operacji, którą z użyciem tego narzędzia można przeprowadzić.

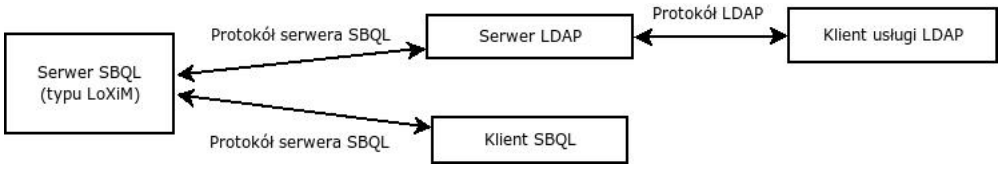
W zastosowaniach rzeczywistych — ograniczenie powyższe prowadzi do przechowywania części istotnych danych z usługi katalogowej w krotkach relacyjnych bazy danych — które to umożliwiają przeprowadzanie złączeń danych należących do różnych encji. W szczególności jeśli mamy strukturę LDAP przechowującą dane osobowe pracowników, wykorzystywana do przeprowadzania autoryzacji do oprogramowania korporacyjnego, to także musimy mieć relacyjną kopię tych danych na potrzeby działania oprogramowania finansowego lub kadrowego. Ubogość tych mechanizmów prowadzi więc do redundancji, a tym samym do dodatkowych

nakładów na utrzymanie tych danych i brak generycznych mechanizmów zapewniających ich spójność.

Wydaje się, że sytuacja byłaby o wiele lepsza, gdyby jednocześnie do danych obsługiwanych przez usługę katalogową istniał dostęp za pomocą języka zapytań o nieporównywalnie większych możliwościach. Wtedy odpowiednia baza danych umożliwiająca dostęp poprzez obydwa interfejsy mogłaby pełnić rolę centralnego źródła danych.

Poniżej przedstawiamy 3 ogólne schematy systemu realizującego tę funkcjonalność.

Serwer SBQL jako zaplecze (ang. backend) usługi LDAP



Rysunek C.1: Schemat serwera SBQL jako zaplecza usługi LDAP

W schemacie tym mamy zwykłą bazę danych realizującą dostęp poprzez język SBQL oraz serwer LDAP wykorzystujący tę bazę danych jako narzędzie do przechowywania i wyszukiwania informacji. Do serwera usługi LDAP jest zapewniony dostęp za pomocą protokołu LDAP. Inne systemy chcące analizować dane przy pomocy SBQL'a powinny dostawać się bezpośrednio do bazy danych SBQL.

Jest to najbardziej naturalne rozwiązanie od strony implementacyjnej i architektonicznej. Istnieją w nim dwa prawie niezależne komponenty skupione na świadczeniu konkretnych usług.

Zastosowanie tego modelu rodzi następujące spostrzeżenia:

- Nie istnieje żadna zależność pomiędzy protokołem dostępu do bazy danych SBQL, a protokołem LDAP.

- Przechowywanie danych poprzez serwery usług katalogowych w semistrukturalnych, obiektowych bazach danych wydaje się być o wiele lepszym rozwiązaniem niż wykorzystywanie do tego celu relacyjnych baz danych (np. oprogramowanie OpenLDAP przechowuje dane w bazie Berkeley DB). Można się spodziewać, że z czasem powstaną narzędzie zorganizowane w ten sposób.

- Opracowanie jednego standardu przechowywania danych usług katalogowych w bazach typu LoXiM — zanim powstaną różne realizacje usług opartych na tym modelu — umożliwi łatwiejszą wymianę konkretnej implementacji usługi LDAP lub budowania wielu programów korzystających z drzewa typu LDAP poprzez język SBQL. Dlatego w rozdziale „Symulowanie modelu danych protokołu LDAP za pomocą modelu danych SBQL" (rozdział C.2.2) spróbujemy przeanalizować dostępne możliwości w tej kwestii.

- Istnieje ryzyko, że modyfikacje danych przeprowadzane za pomocą języka SBQL będą naruszały warunki stawiane strukturze danych serwera SBQL symulującej katalog SBQL. Dlatego wskazane jest by taki schemat danych był kontrolowalny pod względem spójności za pomocą mechanizmów serwera SBQL.

Rysunek C.2: Schemat serwera LDAP z dostępem przez SBQL

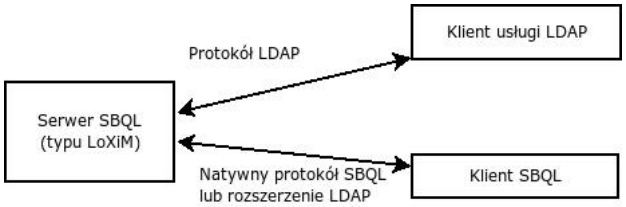
Serwer LDAP świadczący dostęp za pomocą języka SBQL

Alternatywnym scenariuszem jest sytuacja, w której obecne rozwiązania LDAP zaczęłyby być rozszerzane do obsługi języka typu SBQL. Wtedy naturalnym rozwiązaniem jest wprowadzenie do protokołu LDAP rozszerzenia (nowych typów pakietów), które umożliwią zadawanie zapytań w języku SBQL, zamiast implementowania na serwerze LDAP zupełnie innego protokołu.

W realizacji tego rozwiązania będzie potrzebny standard mówiący jakiej strukturze danych serwera SBQL odpowiada dany katalog LDAP, a więc opisujący w jaki sposób wykonywać zapytania SBQL. Problem ten został rozwinięty w rozdziale C.2.2 „Symulowanie modelu danych protokołu LDAP za pomocą modelu danych SBQL".

Drugim brakującym elementem jest specyfikacja rozszerzenia protokołu LDAP, umożliwiająca w tym protokole wykonywanie zapytań SBQL. Problemem tym zajmujemy się w rozdziale C.2.4.

Serwer SBQL świadczący dostęp do danych za pomocą protokołu LDAP



Rysunek C.3: Schemat serwera SBQL świadczącego dostęp do danych za pomocą protokołu LDAP

W tym modelu rozpatrujemy serwer SBQL, który nie tylko obsługuje dostęp za pomocą języka SBQL, ale także udostępnia zasoby za pomocą mechanizmów protokołu LDAP. Gdyby możliwe było sensowne wykonywanie wyszukiwań LDAP oraz poleceń modyfikujących dane protokołu LDAP na każdym modelu danych serwera SBQL (modelu AS_0 lub wyższego) to realizacja tego modelu miała by sens. Umożliwiłaby ona wykorzystanie już istniejących rozwiązań integrujących się z usługami katalogowymi i operowanie bezpośrednio na bazie typu LoXiM.

Kwestiami wykonywania operacji protokołu LDAP na ogólnym modelu danych serwera SBQL (AS_0) zajmiemy się szczegółowo w rozdziale C.2.3.

By nie mnożyć bytów w tym rozwiązaniu miałoby sens zastosowanie jednego protokołu dostępu — którym byłby protokół LDAP wraz z pewnymi rozszerzeniami umożliwiającymi wykorzystywanie języka SBQL do przeprowadzania zapytań. Są to te same rozszerzenia, które

dotyczą „Serwera LDAP świadczącego dostęp za pomocą języka SBQL" i które szczegółowo omawiamy w rozdziale C.2.4.

C.2. Analiza

C.2.1. Porównanie modeli danych

SBQL

Rozważmy najbardziej ogólny model danych dla języka SBQL — (AS_0 Store Model). W modelu tym obiekty są trójkami <identyfikator,nazwa,wartość>, w których wyróżniamy w trzy przypadki:

Obiekty atomowe (proste) — (ang. atomic objects) <identyfikator,nazwa,wartość prosta>

Obiekty wskaźnikowe — (ang. pointer objects) <identyfikator,nazwa,identyfikator docelowy>

Obiekty złożone — (ang. complex objects) <identyfikator,nazwa,kolekcja obiektów> Gdzie kolekcja obiektów może być zbiorem (ang. set) bądź sekwencją (ang. sequence) elementów (w modelu $AS_{0_{seq}}$)

Model ten narzuca dodatkowo pewne wymagania dotyczące "zgodności" danych:

- Unikatowość identyfikatorów obiektu. W całym systemie nie istnieją dwa elementy o takim samym identyfikatorze (pierwszym elemencie trójki)
- Jeżeli obiekt jest typu wskaźnikowego, to obiekt na który wskazuje musi istnieć

Dodatkowo obowiązuje założenie całkowicie ukrytego identyfikatora (Total internal object identification) w „Principles of query programming languages" [SBQL], które oznacza, że użytkownik (aplikacja kliencka) nie może poznać identyfikatora obiektu z którym pracuje. Zatem w tym modelu danych nie istnieje żaden zewnętrzny identyfikator (adres), który by potrafił unikatowo zidentyfikować konkretny obiekt w całej bazie danych implementującej ten model.

Model danych LDAP

Dane są reprezentowane w postaci hierarchii obiektów (wpisów) (ang. entries). Szczytowe (ang. top) obiekty takiego drzewa nazywane są korzeniami (ang. roots/base/suffixs). Każdy obiekt ma maksymalnie jednego ojca i może mieć dowolną liczbę dzieci oraz zbiór atrybutów.

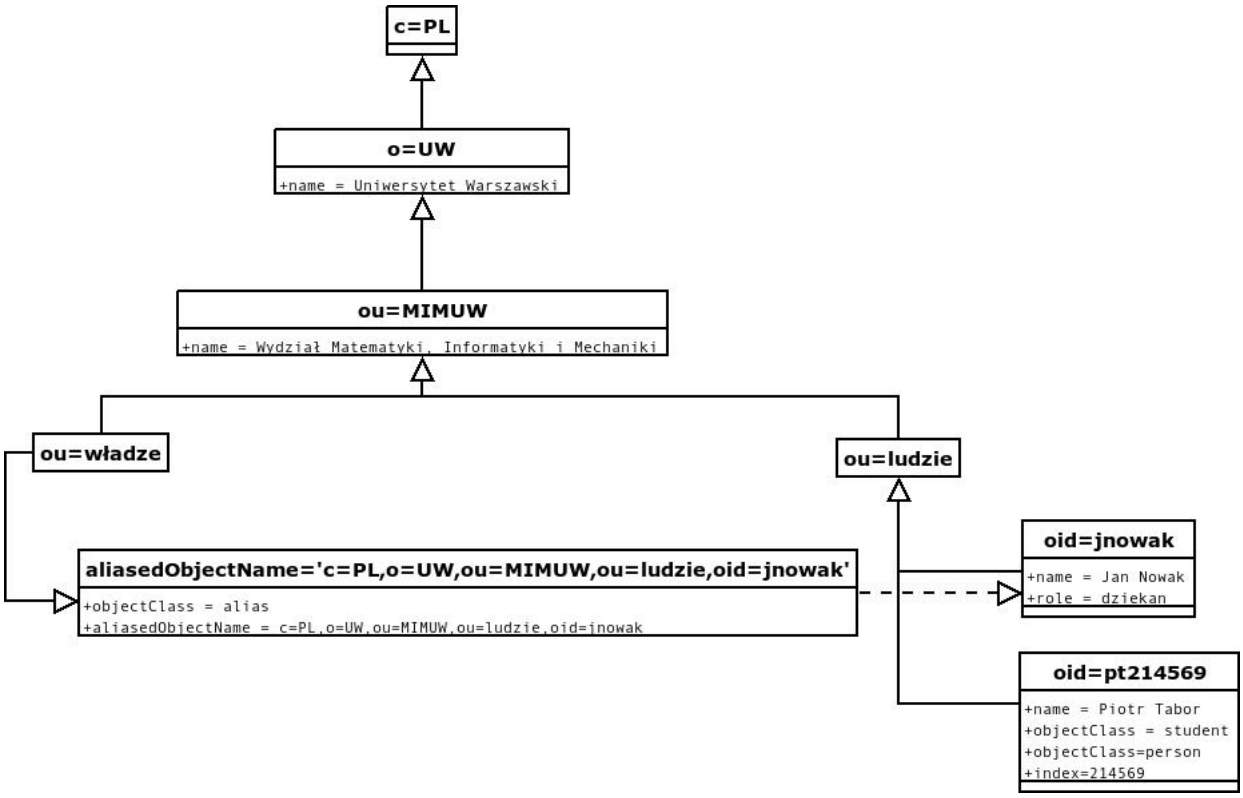
Każdy atrybut ma nazwę i może mieć jedną lub wiele wartości. Wartości w obrębie pojedynczego atrybutu nie mogą być tożsame (definicja tożsamości zależy od atrybutu (matching rule). Kolejność wartości w obrębie atrybutu o wielu wartościach nie ma znaczenia. Można myśleć o atrybucie z wieloma wartościami jak o wielu atrybutach z taką samą nazwą i różnymi wartościami.

Dla różnych atrybutów może być w różnych sposób zdefiniowana identyczność (matching rule). W szczególności specyfikacja, czy porównanie jest zależne od wielkości znaków (case sensitive/insensitive) jest najczęściej używaną własnością atrybutu.

Obiekty od danego korzenia budują drzewo DIT (Directory Information Tree). W obrębie drzewa każdy obiekt można zaadresować za pomocą DN (Distinguish Name). DN buduje się jako ciąg obiektów par: (atrybut=wartość), specyfikujących pełną ścieżkę od obiektu do korzenia.

Np. cn=Piotr Tabor,ou=MIMUW,o=UW,city=Warszawa,country=Polska.

Protokół LDAP przewiduje istnienie aliasów, które polegają na tym, że element może pokazywać na Distinguish Name innego elementu. Z punktu widzenia przeszukiwania (o ile flaga dereferencji aliasów w zapytaniu jest włączona) taki element jest traktowany jakby należał do poddrzewa. Linki te można rozumieć jako analogię do dowiązań symbolicznych w systemach Uniksowych.



Rysunek C.4: Przekład 1 danych w modelu LDAP

C.2.2. Symulowanie modelu danych protokołu LDAP za pomocą modelu danych SBQL

Podejście 1

Na podstawie danych w modelu LDAP można zbudować ich instancję w modelu AS_0 w następujący rekurencyjny sposób (załóżmy, że mamy zbudować strukturę dla poddrzewa zaczynającego się w danym wpisie E):

Tworzymy obiekt okalający o id (i_1), zawierające następujący wiązania (bindings):

entry — obiekt (wpis) właściwy.

- Dla każdego atrybutu należącego do wpisu E tworzymy wiązanie o nazwie atrybutu prowadzące do wartości tego atrybutu.
- Dla atrybutu kluczowego (budującego „distinguish name") tworzymy wiązanie o nazwie atrybutu poprzedzonej '#' prowadzące do wartości atrybutu kluczowego.
- Dla każdego wpisu bezpośrednio podległego tworzymy binding o nazwie '#child#' prowadzący do rekurencyjnie wygenerowanego dla tego wpisu obiektu okalającego.

deref - Jeżeli bieżący wpis jest aliasem to „deref" wskazuje na cel aliasu (bezpośrednio, a nie na obiekt okalający). W przeciwnym przypadku wiąże do tego samego obiektu co „entry".

Przykład z rysunku przedstawionym w "Modelu danych LDAP", zapisany w AS_0 z wykorzystaniem tego podejścia (po dodaniu nadrzędnych obiektów root i #child#):

```
1 root:
  2 #child#:
     3 deref: ->5
     5 entry:
        6 c=PL
        7 #c -> 6
        8 #child#:
           9 deref: ->10
           10 entry:
              11 ou=UW
              12 #ou -> 11
              13 name=Uniwersystet Warszawski
              14 #child#:
                 15 deref: ->16
                 16 entry:
                    17 ou=MIMUW
                    18 #ou=MIMUW
                    19 name=Wydział Matematyki, Informatyki i Mechaniki
                    20 #child#
                       21 defer: ->22
                       22 entry:
                          23 ou=ludzie
                          24 #ou ->23
                          25 #child#
                             26 deref ->27
                             27 entry:
                                28: oid=jnowak
                                29: #oid ->28
                                30: name=Jan Nowak
                                31: objectClass: person
                                50: role: dziekan
                          32 #child#
                             33 deref ->34
                             34 entry:
                                35: oid=pt214569
                                36: #oid ->35
                                37: name=Piotr Tabor
                                38: index=214569
                                39: objectClass: person
                                40: objectClass: student
                    41 #child#
                       42 deref: ->43
                       43 entry:
```

```
44 ou=Władze
45 #ou ->44
46 #child#:
   47 deref ->27
   48 entry:
      49 aliasedObjectName='c=PL,o=UW,ou=MIMUW,ou=ludzie,oid=jnow
      51 #aliasedObjectName ->49
      52 objectClass: alias
```

W tym modelu możemy stosunkowo łatwo przetłumaczyć „distinguish name" na zapytanie odnajdujące wskazany adres. Następujący adres: c=PL,o=UW,ou=MIMUW,ou=ludzie,oid=jnowak zostanie przetłumaczony na:

W przypadku nie rozwijania aliasów :
(root).(entry where #c=PL).#child#.(entry where #ou=UW).#child#
.(entry where #ou=MIMUW).#child#.(entry where #ou=ludzie).#child#
.(entry where #oid=pt214569).#child#

W przypadku rozwijania aliasów :
(root).(deref where #c=PL).#child#.(deref where #ou=UW) .#child#
.(deref where #ou=MIMUW).#child#.(deref where #ou=ludzie).#child#
.(deref where #oid=pt214569).#child#

C.2.3. Operacje protokołu LDAP

Operacje protokołu StartTLS

Wysłanie paczki StartTLS przez klienta protokołu LDAP powoduje rozpoczęcie negocjacji TLS (Transport Layer Security), czyli bezpiecznego — szyfrowanego — połączenia. Operacja ta jest istotna i w oczywisty sposób łatwa do zaimplementowania w przypadku wykorzystania protokołu do komunikacji z innym rodzajem bazy danych.

Bind — autoryzacja

W protokole LDAP klient przeprowadzając autoryzację przekazuje następujące parametry serwerowi:

wersje protokołu — po której chce się komunikować (obecnie dozwolona tylko wersja 3)

nazwa (DN) logującego się — adres (DN) wpisu identyfikującego obiekt autoryzujący się (przeważnie użytkownika)

dane autentykujące — paczka danych opisujących wybraną metodę autentykacji oraz zestaw danych potrzebnych do jej przeprowadzenia. Specyfikacja podstawowych metod znajduje się w dokumencie [RFC5420] i przewiduje: logowanie anonimowe, logowanie z podaniem loginu i hasła (czystego bądź zaszyfrowanego) oraz usługi SASL (Simple Authentication and Security Layer).

Wykorzystanie tej metody w modelu bazy semistrukturalnej niesie ze sobą następujące trudności:

- Zakłada, że użytkownicy bazy danych są dostępni jako element struktury danych. To założenie wydaje się sensowne. W najgorszym przypadku użytkownicy mogą być udostępniani jako fragment wirtualnego modelu danych (analogia do katalogu /proc w UNIX'ach).

- Zakłada, że dysponujemy mechanizmem tłumaczenia adresów DN na konkretne obiekty modelu AS_0 ((patrz: C.2.2)) (w ogólnym przypadku danych - niemożliwe lub bardzo siłowe)

Drugie z wymieniony założeń jest problemem trudnym, ale przyjmując nawet, że jest nierozwiązywalny z niewielką stratą możemy przyjąć, że operacja BIND z atrybutem name="o=loxim,cn=kowalski" odpowiada próbie autentykacji użytkownika Kowalski do bazy danych.

Reasumując — pakiet ten jest istotny i możliwe jest jego wykorzystanie przy dostępie do bazy danych typu LoXiM.

Unbind — zakończenie połączenia

Nazwa tego polecenia jest myląca, gdyż w protokole LDAP oznacza ona operacje zamknięcia połączenia (a nie — jak mogła by sugerować — wylogowania użytkownika).

Operacja nie niesie ze sobą żadnych trudności w dostępie do danych o semistrukturalnym modelu.

Search — wyszukiwanie

Operacja „search" jest najważniejszą operacją w usługach katalogowych. Przyjmuje następujące parametry:

baseObject — adres (DN) miejsca w drzewie od którego (w głąb) rozpoczynamy przeszukiwanie

scope — zakres drzewa, który chcemy przeszukać:

- **baseObject** — przeszukany zostaje tylko wskazany obiekt (w praktyce oznacza to sprawdzenie, czy wskazany obiekt spełnia zadane warunki lub pobranie wybranych atrybutów z obiektu)
- **singleLevel** — przeszukane zostają (bezpośrednie) dzieci danego obiektu
- **wholeSubtree** — przeszukane zostaje całe poddrzewo wyznaczone przez „baseObject", czyli wszyscy potomkowie tego obiektu z nim włącznie

derefAliases — jak aliasy (odpowiednik obiektów pointerowych) mają być traktowane:

- **neverDerefAliases** — nigdy nie interpretuj aliasów
- **derefInSearching** — interpretuj aliasy tylko w trakcie przeszukiwania (ale nie w trakcie znajdowania „korzenia przeszukiwania" (baseObject))
- **derefFindingBaseObj** — interpretuj aliasy w trakcie znajdowania „korzenia przeszukiwania" (baseObject)
- **derefAlways** — zawsze interpretuj aliasy

sizeLimit — maksymalna liczba zwróconych wpisów

timeLimit — maksymalny czas przeszukiwania (w sekundach). W przypadku przekroczenia — zwrócone zostaną wpisy znalezione do czasu jego upłynięcia.

typesOnly — true lub false — decyduje, czy zapytanie zwraca pary: opis atrybutu i jego wartość, czy tylko opisy atrybutów

filters — złożony obiekt reprezentujący warunki jakie muszą spełniać zwrócone wpisy (obejmuje: porównania atrybutów ze stałymi (także przybliżone), operacje logiczne, sprawdzanie istnienia atrybutu).

attributes — lista atrybutów, które mają zostać zwrócone dla każdego znalezionego wpisu. Można też przekazać wartość żądającą wszystkich atrybutów zawartych w danym wpisie (bez ukrytych).

W odpowiedzi na tę operację będą przychodzić (asynchronicznie): lista znalezionych obiektów z wartościami wskazanych atrybutów — a także lista serwerów, które mogą znać więcej odpowiedzi na to zapytanie (może być warto przeprowadzić to samo przeszukiwanie na nich, ale to decyzja klienta).

Wykorzystanie tego mechanizmu do przeszukiwania bazy danych opartej na języku SBQL wymaga:

- przetłumaczenia adresu „baseObject" na konkretny obiekt modelu AS_0 (czyli na zapytanie SBQL znajdujące ten obiekt) ((patrz: C.2.2)) z interpretacją (lub nie) aliasów — w zależności od wartości parametru „derefAliases" (w ogólnym przypadku danych — niemożliwe lub bardzo siłowe)
- Przetłumaczenie warunków wyszukiwania zadanych parametrem „filter" na SBQL'a — uwzględniającego wybrany „scope", a także rozwijanie aliasów („deref aliases"). Zakładając, że możemy składnie SBQL'a rozszerzyć o funkcje dokonujące bardziej złożonych porównań (np. z uwzględnieniem skrótów MD5 lub SHA1) co wydaje się w pełni realizowalne — o ile wiemy jak interpretować pojęcie „wpis" i „atrybut" w kontekście przetwarzanych danych (AS_0).

Problemy te — jeśli zakładamy, że baza danych zawiera dane w ogólnym modelu AS_0 wydają się uniemożliwiać przeprowadzenie tej operacji. Jednak na danych, które potrafimy zinterpretować jako „wpisy" i „atrybuty" możemy z powodzeniem i wydajnie zaimplementować tę operację.

Operacja ta nie nadaje się do zadawania zapytań w języku SBQL w szczególności ze względu na stosunkowo „płaski" format odpowiedzi, a także braku parametru umożliwiającego wygodne przekazanie zapytania SBQL wraz z wartościami jego parametrów. Z tego względu — jeśli chcemy zapewnić możliwość zadawania zapytań SBQL bazie danych — musimy zaimplementować nową operację do tego przeznaczoną (rozszerzenie protokołu LDAP).

Modify — zmiana wpisu

Operacja „modify" umożliwia klientowi dodanie lub usunięcie atrybutów z wpisu, a także zamianę wartości wskazanych atrybutów. Zawiera dwa parametry:

object — adres (DN) wpisu, który chcemy zmodyfikować.

changes — listę operacji modyfikacji. Operacjami modyfikacji może być:

add — dodanie atrybutu (lub wartości do atrybutu — jeśli ten już istnieje)

delete — usunięcie całego atrybutu (lub pojedynczej wartości — jeśli została ona wskazana)

replace — zamienienie wszystkich wartości wskazanego atrybuty na załączone do tej operacji.

Przeprowadzenie tej operacji w bazie typu SBQL jest możliwe w sytuacji kiedy umiemy mapować pojęcia „wpis" i „atrybut" na model danych. W ogólnym przypadku musimy odnaleźć „object" (czyli przetłumaczyć DN na zapytanie SBQL), a następnie w zależności od wybranej operacji musielibyśmy usunąć, zmodyfikować lub dodać nowe dziecko typu „binder" do wskazanego obiektu. Tworzony binder będzie związany z wartością typu prostego.

Add — dodanie wpisu

Operacja przyjmuje dwa parametry: adres (DN) wpisu, który chcemy utworzyć oraz listę atrybutów wraz z wartościami dla tego obiektu.

Przy założeniu, że umiemy przetłumaczyć DN na obiekt modelu AS_0, który chcemy utworzyć — operację tę możemy wykonać przez utworzenie prostego obiektu typu „binder" dla każdego atrybutu we wskazanym obiekcie.

Delete — usunięcie wpisu

Operacja usuwa wskazany wpis. Jej jedynym parametrem jest DN wpisu, który chcemy usunąć.

Przy założeniu, że umiemy przetłumaczyć DN na obiekt modelu AS_0, który chcemy usunąć — ta operacja jest wykonywalna i istotna.

Abandon — przerwanie przetwarzanej właśnie operacji

Operacja ta przyjmuje parametr będący id wykonywanej aktualnie operacji (przeważnie polecenia SEARCH) i wymusza jej zakończenie.

Operacja istotna i możliwa do implementacji w bazie danych opartej na SBQL.

C.2.4. Podsumowanie

Pomimo wielu cech wspólnych obu narzędzi (patrz: C.1.4) wykorzystanie czystego protokołu LDAP jako jedynego interfejsu komunikacyjnego jest problematyczne.

Wykazaliśmy, że operacje „bind", „unbind", „startTLS", „abandon" można stosunkowo łatwo wykorzystać w bazie danych typu SBQL (można je wykorzystać w każdym protokole, który przeprowadza autentykacje). Stanowią one „standardowy" szkielet, na którym można budować protokół właściwy.

Operacje „add", ‚delete" i „modify" jesteśmy w stanie przeprowadzić w bazie danych typu LoXiM o ile potrafimy zinterpretować DN jako adres konkretnego obiektu w tej bazie. W praktyce jest to tylko możliwe jeśli baza używa schematu, który jesteśmy wstanie zinterpretować jako zgodny z usługą LDAP (patrz: C.2.2).

Operacja „search" jest także tylko możliwa, gdy dane bazy danych potrafimy zobrazować jako zgodne z usługą LDAP.

Do przeprowadzania zapytań i modyfikacji przy pomocy języka SBQL będziemy potrzebowali wprowadzić rozszerzenie protokołu LDAP (wprowadzanie rozszerzeń jest przewidziane przez ten standard).

Wizja realizacji

W tym podpunkcie przedstawię pomysł na maksymalną integrację bazy danych SBQL i protokołu LDAP, która ma sens:

1. Dostęp do takiej bazy danych byłby realizowany przez protokół LDAP z dodatkowymi pakietami umożliwiającymi wykonywanie parametryzowanych zapytań SBQL.
2. W schemacie bazy danych powinny istnieć wyróżnione poddrzewa, który mają określoną strukturę — zgodną ze schematem usługi LDAP (najlepiej kontrolowaną poprzez model np. AS_1 lub wyższe). Dostęp do tych danych (i tylko nich) powinien być zapewniony poprzez operacje „Search", „Add" , „Delete" i „Modify".
3. Do całego schematu bazy danych powinien istnieć dostęp za pomocą języka SBQL przy pomocy rozszerzenia protokołu omówionego w punkcie 1.

Takie rozwiązanie ma następujące zalety:

- Ułatwiamy życie programistom narzędzi klienckich — którzy mogą wykorzystać już istniejący (dla protokołu LDAP) kod do autentykacji.
- Korzystamy z bogactwa metod autentykacji przygotowanych już dla protokołu LDAP.
- Ułatwiamy integrację wszystkich danych w pojedynczym narzędziu ze spójnym interfejsem.
- Ułatwiamy migrację ze standardowego modelu „LDAP + baza danych" do modelu „SBQL z interfejsem LDAP".
- Nie tworzymy „nowych (wcale nie lepszych) standardów" w informatyce.
- Umożliwiamy współpracę bazy typu LoXiM z wieloma już istniejącymi aplikacjami i bibliotekami.

Dodatek D

ProtoGen 1.0 - dokumentacja generatora protokołów sieciowych

D.1. Wprowadzenie

D.1.1. Wstęp

Generator protokołów jest narzędziem służącym do generowania implementacji protokołu sieciowego w różnych językach programowania w oparciu o zadany opis. Danymi wejściowymi generatora jest specyfikacja protokołu zapisana w pliku XML (patrz: D.2.2) oraz wybrany język programowania. Danymi wyjściowymi jest kod źródłowy w wybranym języku programowania umożliwiający wygodne posługiwanie się tym protokołem.

D.1.2. Licencjonowanie

Projekt „ProtoGen 1.0" jak i wygenerowany przez niego kod jest licencjonowany na zasadach licencji „Apache Software License 2.0" (http://www.apache.org/licenses/LICENSE-2.0).

D.1.3. Korzyści płynące ze stosowania tego rozwiązania

Zastosowanie tego narzędzia daje następujące korzyści:

- Gwarantuje zgodność na poziomie danych przysyłanych siecią dla implementacji w różnych językach programowania
- Pozwala uniknąć programiście „mechanicznego" tworzenia dużych ilości kodu a zaraz błędów z tym związanych
- Gwarantuje, że cały kod protokołu jest napisany w sposób spójny
- Gwarantuje, że wygenerowany kod jest odporny na podstawowe zagrożenia w komunikacji pomiędzy różnymi architekturami: problem kolejności bajtów w słowie (big-endian/little-endian) oraz problem rozmiaru podstawowych typów danych (architektury od 8 do 64 bitów).
- Tworzy zestawy testów, które umożliwiają sprawdzanie komunikacji także pomiędzy kodem wygenerowanym dla różnych języków programowania.
- Umożliwia aktualizację całego modułu poprzez aktualizację generatora i ponowne wygenerowanie kodu.

D.1.4. Ograniczenia rozwiązania

ProtoGen nie jest narzędziem, które umożliwia wygenerowanie modułu obsługującego każdy protokół. W obecnej wersji zostały przyjęte poniższe ograniczenia.

Budowa paczki

Każda paczka ma następujący schemat budowy:

Od - do	Typ/Wartość	Zawartość
$0 \rightarrow a$	'zależy od konfiguracji'	Stała mówiąca o typie paczki, a tym samym określająca format zawartych w nich danych
$a+1 \rightarrow a+4$	uint32	n- Stała określająca ilość danych właściwych zawartych w paczce - wyrażona w bajtach
$a{+}5 \rightarrow a{+}5{+}n$	patrz opis zależny od typu paczki	Dane właściwe paczki zgodne z formatem określonym poprzez typ paczki

Formalnie będziemy mówili, że paczka się składa z dwu-polowego nagłówka oraz ciała. Nagłówek wyznacza typ paczki i rozmiar danych właściwych w niej zawartych, a ciało - to dane interpretowane zależnie od typu paczki.

Big/endian - little/endian

Przyjęto, że wszystkie dane są przesyłane przez sieć w formacie big-endian.

Format napisów

Przyjęto, że wszystkie napisy są przesyłane przez sieć w formacie UTF-8.

D.1.5. Złożona logika

Obecnie ProtoGen nie wspiera złożonej logiki w konstrukcji pakietów. Niemożliwe jest warunkowanie istnienia pola w zależności od wartości innego pola, a także tworzenia tablicy pól. Obecnie ProtoGen przewiduje możliwość nadpisania wygenerowanego kodu. W ten sposób - nadpisując kod obsługi odpowiedniej paczki - można uzyskać obsługę dowolnie złożonej logiki.

O planowanym rozwiązaniu tego problemu w przyszłych wersjach możesz przeczytać w rozdziale „Dalszy rozwój” (patrz: D.4.3)

D.2. Dokumentacja użytkownika

D.2.1. Uruchomienie

ProtoGen uruchamiamy będąc w katalogu w którym znajdują się jego pliku uruchomieniowe. W zależności od systemu operacyjnego uruchamiamy protoGen.sh (UNIXy), bądź protoGen.bat (MS Windows) podając następujące parametry:

ścieżka do deskryptora - ścieżka do deskryptora xml opisującego protokół (może być względem bieżącego katalogu). Wskazany plik powinien być w formacie opisanym w rozdziale D.2.2.

ścieżka do katalogu docelowego - ścieżka do katalogu w którym będą umieszczane wygenerowane pliki.

język docelowy - język do którego generujemy kod. Obecnie obsługiwane wartości tego parametru to: „cpp" dla C++ oraz „java".

ścieżka do katalogu z plikami nadpisującymi (parametr opcjonalny) - parametr służy do wskazania katalogu w którym powinny się znajdować pliki, którymi chcemy nadpisać działanie generatora protokołu. Może się zdarzyć sytuacja w której protoGen generowałby plik „ABC.xxx". Jeżeli we wskazanym katalogu będzie się znajdował plik „ABC.xxx" to on właśnie zostanie użyty zamiast wygenerowanego pliku. Pliki umieszczamy w tym katalogu bezpośrednio (bez podkatalogów).

Konieczność nadpisania pliku pojawia się najczęściej w sytuacji, gdy mamy paczkę zawierającą skomplikowaną logikę. Wtedy piszemy kod obsługi takiej paczki ręcznie, a następnie zmuszamy protoGen, by go wykorzystał (umieścił w kodzie wynikowym).

Przykład

Polecenie:

```
./protoGen.sh ./conf/loxim.xml ./cpp_proto cpp ./conf/overwritten/cpp
```

Spowoduje wygenerowanie kodu w języku C++ w podkatalogu cpp_proto, w oparciu o deskryptor ./conf/loxim.xml - sprawdzając, czy istnieją nadpisujące pliki w katalogu ./conf/overwritten/cpp.

D.2.2. Format deskryptora protokołu

Poniżej znajduje się przykładowy deskryptor z opisem poszczególnych znaczników.

Przykładowy deskryptor

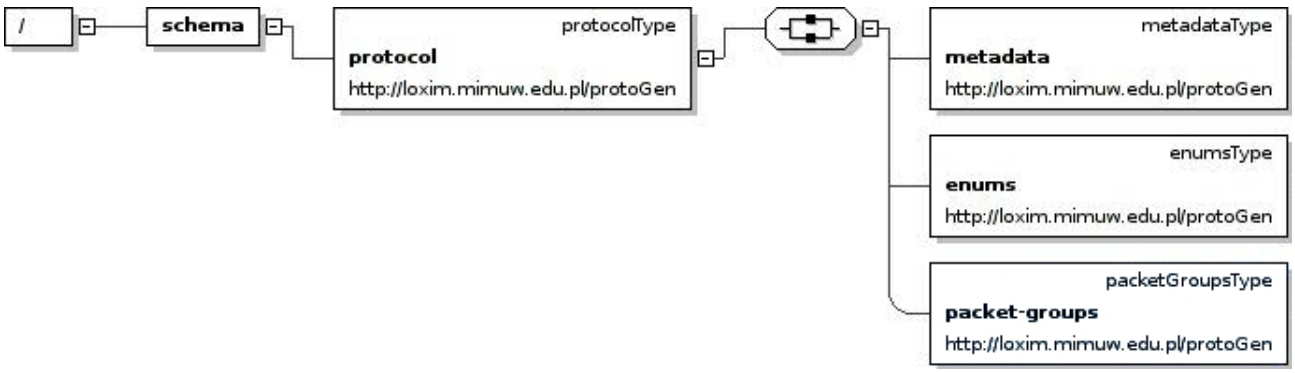
```
<?xml version="1.0" encoding="iso-8859-2"?>
<protocol xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xmlns='http://piotr.tabor.waw.pl/soft/protogen'
    xsi:schemaLocation='http://piotr.tabor.waw.pl/soft/protogen
    http://piotr.tabor.waw.pl/soft/protogen/schema.xsd'>

    <metadata>
```

```
    <major-version>2</major-version>
    <minor-version>0</minor-version>
    <languages>
        <lang id="java">
            <packageName>pl.edu.mimuw.loxim.protocol</
                packageName>
            <artifactId>loxim_protocol</artifactId>
            <version>1.0-SNAPSHOT</version>
        </lang>
    </languages>
</metadata>
<enums>
    <enum name="features" as-type="uint64">
        <item name="f_ssl" value="0x0001">
            <description lang="pl">
                Połaczenie może być szyfrowane metoda SSL
            </description>
        </item>
        ...
    </enum>
    ...
</enums>
<packet-groups>
    <packet-group name="data" id-type="uint32">
        <packets>
            <packet name="uint8"     id-value="1">
                <field name="value" type="uint8"/>
            </packet>
            <packet name="date"    id-value="10">
                <field name="year"  type="sint16"/>
                <field name="month" type="uint8"/>
                <field name="day"   type="uint8"/>
            </packet>
            ...
            <packet name="datetime" id-value="12">
                <field name="date"   type="package" object-ref=
                    "data" object-ref-id="date"/>
                <field name="time"   type="package" object-ref=
                    "data" object-ref-id="time"/>
            </packet>
            <packet name="binding"     id-value="130">
                <field name="bindingName" type="sstring" />
                <field name="type" type="varuint"/>
                <field name="value" type="package-map" object-
                    ref="data" object-ref-id="type"/>
            </packet>
            ...
        </packets>
    </packet-group>
```

```
        <packet-group id-type="uint8">
            <packets>
                <packet name="w_s_hello" id-value="11">
                    <field name="protocol_major" type="uint8">
                        <description lang="pl">Numer główny (major)
                            wersji protokołu</description>
                    </field>
                    <field name="features" type="enum-map" object-
                        ref="features">
                        <description lang="pl">Mapa bitowa
                            dostępnych cech serwera</description>
                    </field>
                    <field name="salt" type="sstring" size="20">
                        <description lang="pl">160 bitowy ciąg
                            losowy - używany przez niektóre
                            metody autoryzacji </description>
                    </field>
                    ...
                </packet>
                <packet name="v_sc_sendvalue" id-value="33">
                    <field name="value_id" type="varuint" />
                    <field name="flags" type="enum-map" object-ref=
                        "send_value_flags" />
                    <field name="value_type" type="varuint" />
                    <field name="data" type="package-map" object-
                        ref="data" object-ref-id="value_type"/>
                </packet>
                ...
            </packets>
        </packet-group>
    </packet-groups>
</protocol>
```

<**protocol**>



Rysunek D.1: Konstrukcja znacznika <protocol>

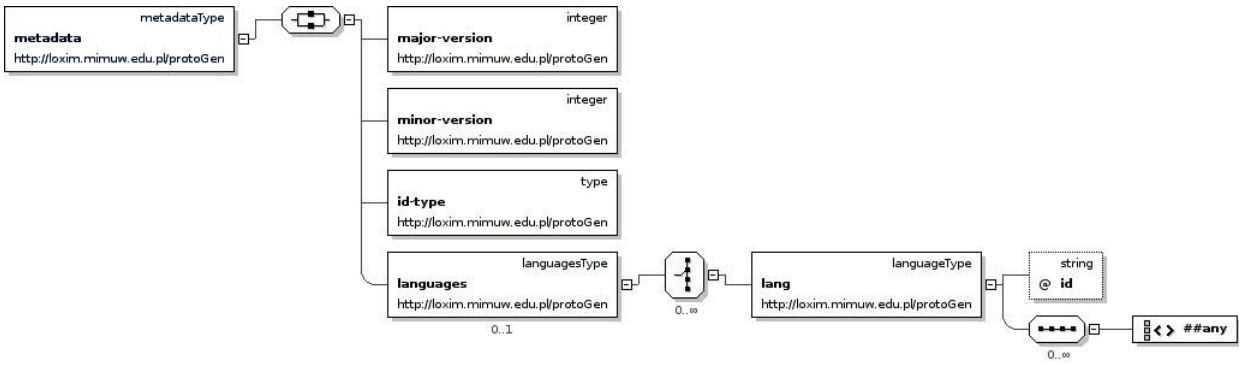
Element <protocol> grupuje znaczniki:

<**metadata**> - opisujące ogólne cechy wygenerowanego protokołu, a także specyficzne cechy zależne od docelowego języka programowania.

<**enums**> - opisuje typy wyliczeniowe, które mogą zostać zastosowane w protokole.

<**packet-groups**> - opisuje grupy paczek (a zatem też paczki), które, bądź budują protokół bezpośrednio, bądź mogą być elementami składowymi paczek.

<metadata>



Rysunek D.2: Konstrukcja znacznika <metadata>

Element <metedata> służy opisaniu pewnych ogólnych własności protokołu.

major-version Główna wersja protokołu opisanego w tym deskryptorze. Przyjmuje się, że protokoły o tym samym głównym numerze wersji są ze sobą kompatybilne (co nie znaczy, że można za pomocą nich przekazać te same informacje, ale to że obie implementację będą rozmawiały tak, jakby implementowały tę samą - starszą - wersję protokołu).

minor-version Poboczny numer wersji protokołu.

languages Definicja cech protokołu zależnych od docelowego języka programowania w którym będziemy generowali kod. Szczegóły opisano poniżej.

<lang id="java">

Znacznik ten opisuje dodatkowe atrybuty potrzebne do wygenerowania protokołu dla języka Java.

packageName
Nazwa pakietu (jako grupy klas w języku Java) zawierającego kod protokołu. Należy oddzielić wszystkie elementy ścieżki przy pomocy symbolu '.'.
Wpisana tu nazwa będzie także <groupId> w utworzonym projekcie Maven2 (http://maven.apache.org).

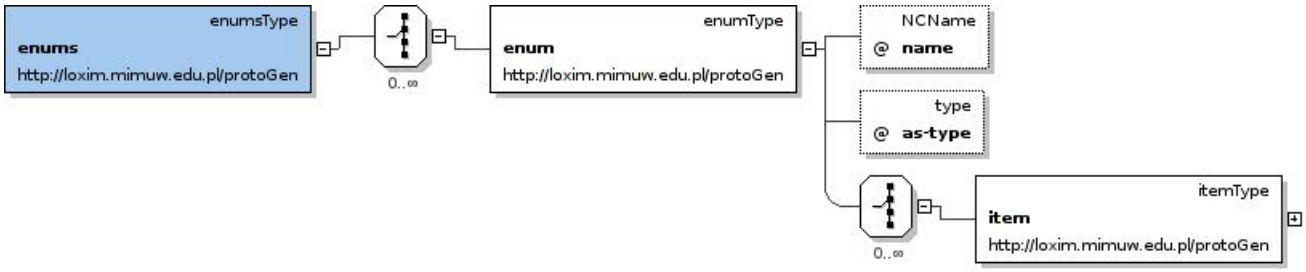
artifactId
Nazwa utworzonego modułu - głównie na potrzeby Maven2.

version
Wersja utworzonego modułu - głównie na potrzeby Maven2.

<lang id="cpp">

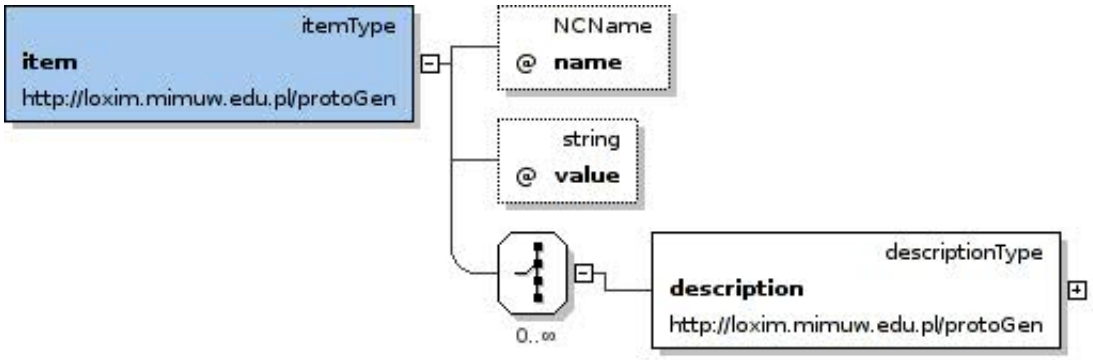
Język C++ nie posiada w obecnej wersji ProtoGen dodatkowych atrybutów.

<enums>



Rysunek D.3: Konstrukcja znacznika <enums>

Znacznik enums opisuje typy wyliczeniowe, które mogą zostać wykorzystane w protokole. Typy wyliczeniowe można wykorzystać bezpośrednio - jako pole zawierające jedną wartość z wielu, a także do stworzenia „mapy wartości", czyli do przechowania zbioru wartości (patrz: D.2.2).



Rysunek D.4: Konstrukcja znacznika <item>

Każdy element (<item>) danego typu wyliczeniowego ma przypisaną wartość typu całkowitoliczbowego. Jeśli chcemy dany typ wyliczeniowy wykorzystać jako „mapę" to musimy nadać poszczególnym elementom wartości o bitach zapalonych rozłącznie.

Znacznik <enum> opisuje pojedynczy typ wyliczeniowy.

Jego atrybuty to:

name Nazwa typu wyliczeniowego. Przy pomocy tej nazwy będzie można się odwoływać do tego typu. Od tej nazwy zależy też nazwa wygenerowanej klasy przechowującej ten typ.

as-type Nazwa typu numerycznego (patrz: D.2.3), na który będą odwzorcowywane poszczególne enumy.

Ponadto znacznik <enum> budują elementy <item>.

<item>

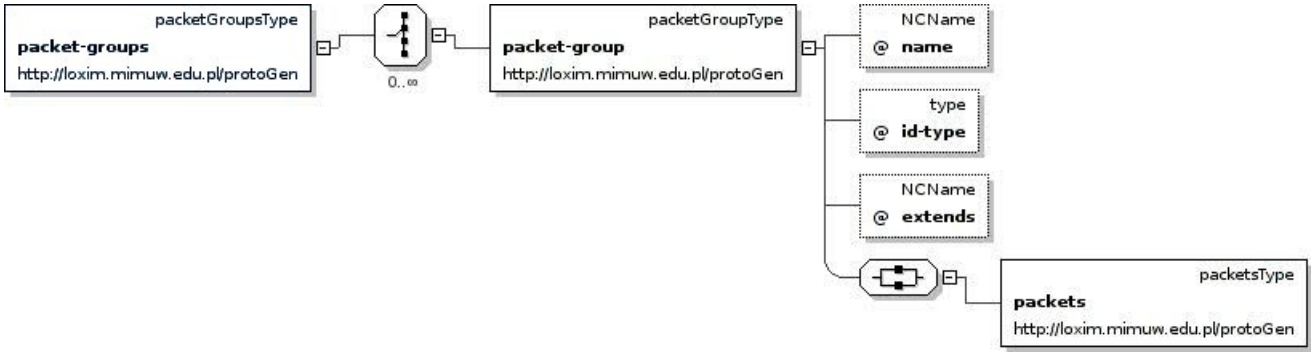
Znacznik item opisuje pojedynczą wartość, którą może przyjąć typ wyliczeniowy. Pojedyncza wartość posiada następujące atrybuty:

name Nazwa elementu typu wyliczeniowego.

value Wartość na którą ten element wyliczeniowy będzie odwzorowywany. Powinna mieścić się w zakresie typu wskazanego w atrybucie „as-type" w znacznika <enum> definiującym ten typ wyliczeniowy. Wartość może być podana w systemie dziesiętnym lub 16-tkowym poprzez poprzedzenie jej symbolami „0x", np. 0x000f4 - co jest szczególnie przydatne przy konstrukcji typów używanych do budowania mapy wartości (patrz: D.2.2).

Ponadto znacznik <item> może posiadać elementy typu <description> zawierające dane do budowania dokumentacji (patrz: D.2.2).

<packet-groups>



Rysunek D.5: Konstrukcja znacznika <packet-groups>

Znacznik <packet-groups> grupuje znaczniki <packet-group>, które stanowią grupę paczek. Grupa paczek to taki zbiór definicji paczek, która ma różne identyfikatory i w podobny sposób jest wykorzystana. W szczególności dla każdej grupy paczek zostanie wygenerowana fabryka paczek, która pozwala powołać do życia paczkę na podstawie zadanego jej „id".

Zawsze powinna istnieć grupa „główna" (bez nazwy). Jej elementy będą stanowiły podstawowe paczki protokołu. Pozostałe (nazwane) grupy są pomocnicze i mogę służyć do budowania pól innych paczek.

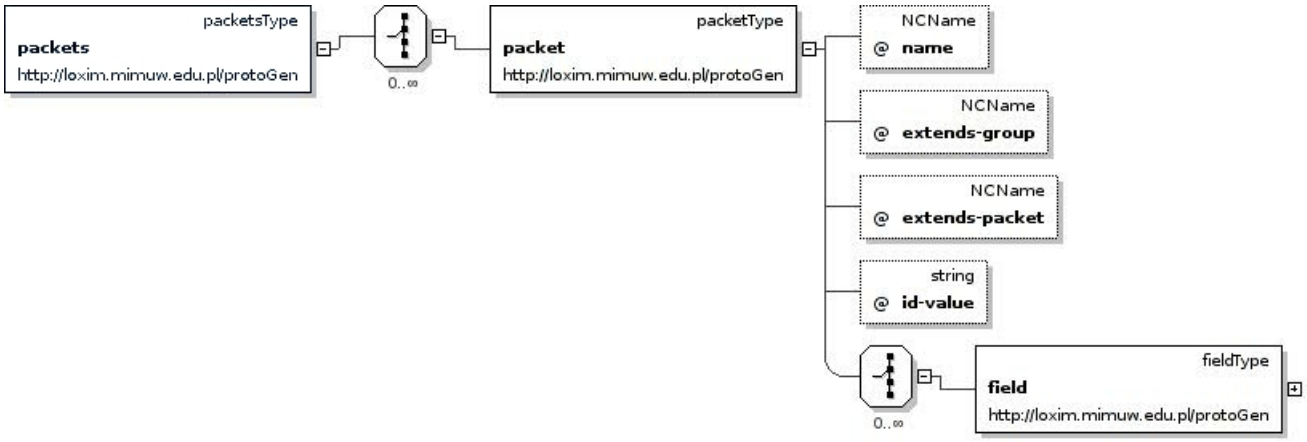
Grupę charakteryzują następujące atrybuty:

name - nazwa grupy. Musi istnieć jedna grupa bez nazwy (grupa „główna")

id-type - nazwa typu całkowitoliczbowego(patrz: D.2.3), którego typu będą identyfikatory paczek.

extend (atrybut opcjonalny) - nazwa paczki, która stanowi bazę dla wszystkich paczek tej grupy (o ile nie nadpisano atrybutem extend paczki). Jeśli nie zdefiniowano to paczki będą dziedziczyły domyślnie z głównego typu „Packet" nie zawierającego żadnych pól.

Znacznik <packet-group> zawiera podelementy <packets>.



Rysunek D.6: Konstrukcja znacznika <packet>

<packets>

Element <packets> grupuje znaczniki <packet>, które opisują pojedynczą paczkę danych. Paczka danych składa się z pól (patrz: D.2.2), a ponadto jest opisana następującymi atrybutami:

name - Nazwa paczki

id-value (atrybut opcjonalny) - Wartość identyfikująca paczkę (typu „id-type" z definicji grupy paczek). Jeśli paczka nie posiada id-value - to jest traktowana jako „abstrakcyjna", co oznacza, że może być wykorzystana tylko jako paczka bazowa dla innych paczek.

extends-group (atrybut opcjonalny) - nazwa grupy paczek, w której się znajduje się paczka zdefiniowana w polu „extends-packet".

extends-packet (atrybut opcjonalny) - nazwa paczki z grupy "extends-group", którą deklarowana właśnie paczka rozszerza (o dodatkowe pola).

<field>

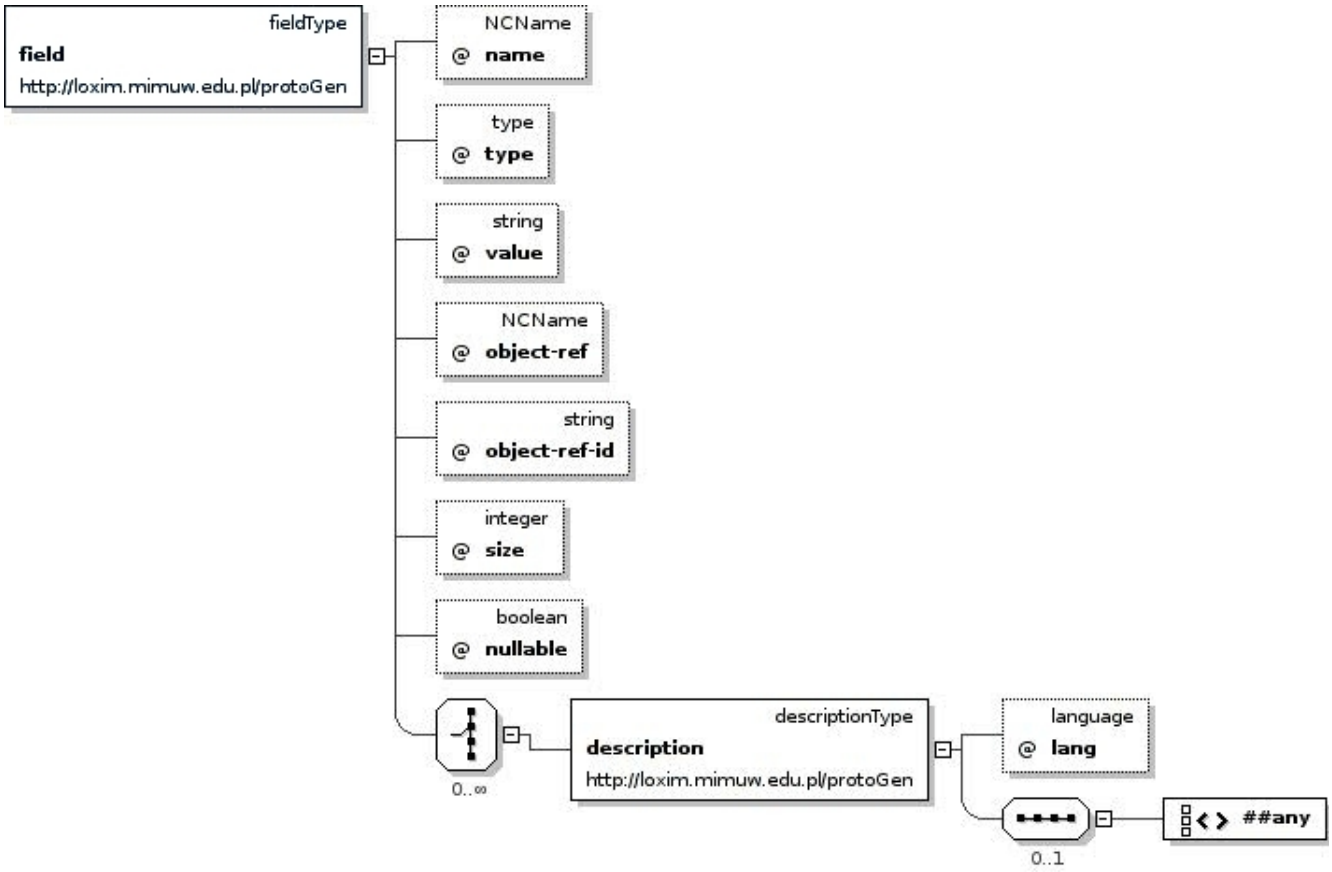
Znacznik <field> opisuje pojedyncze pole paczki. Podelementem znacznika <field> może być znacznik <description> (patrz: D.2.2). Atrybutami znacznika field niezależnie od typu tego pola są:

name - Nazwa pola

type - Typ pola (patrz: D.2.3)

Pozostałe atrybuty zależą od typu pola:

- Dla typów prostych dostępny jest dodatkowo atrybut „value" mogący zawierać domyślną zawartość pola.
- Dla typów: string, sstring, varuint dostępny jest atrybut "nullable" (przyjmujący wartości true lub false), świadczący o tym, czy dopuszczalna jest zawartość NULL dla tego pola.



Rysunek D.7: Konstrukcja znacznika <fields>

- Dla typów: string i sstring dostępny jest atrybut size zawierający maksymalną dopuszczalną długość napisu.

Ponadto następuję typy złożone wymagają szczególnego omówienia:

enum - Pole zawiera jedną z wartości z listy wyboru. Atrybut „object-ref" wskazuję na nazwę odpowiedniego „enuma" (patrz: D.2.2). Pole będzie serializowane jako typ całkowitoliczbowy wskazany w definicji „enuma" o wartości odpowiadającej „id" wybranego elementu z tej listy wyboru.

enum-map - Pole zawiera bitową mapę wartości w listy wyboru. Atrybut „object-ref" wskazuję na nazwę odpowiedniego „enuma" (patrz: D.2.2). Pole będzie serializowane jako typ całkowitoliczbowy wskazany w definicji „enuma" o wartości odpowiadającej sumie „id" wybranych elementu z tej listy wyboru.

package - Pole zawiera ciało (bez nagłówka) całej innej paczki. Atrybut „object-ref" wskazuję na nazwę grupy paczek, a „object-ref-id" na nazwę konkretnej paczki należącej do tej grupy.

package-map - Pole zawiera ciało całej innej paczki należącej do wskazanej grupy paczek, ale konkretny rodzaj paczki jest wyznaczony przez wartość innego - uprzednio wymienionego pola. Zatem „object-ref" wskazuję na nazwę grupy paczek, a „object-ref-id" na

nazwę wcześniej zdefiniowanego pola w bieżącej paczce - które będzie zawierało identyfikator paczki, który ma zostać w bieżącym polu wczytany/zapisany. O polu „package-map" należy myśleć jak o polu „package" z dynamicznie wyznaczanym konkretnym rodzajem paczki.

<description>

Znacznik <description> może występować w wielu miejscach specyfikacji protokołu. Służy on do wygenerowania dokumentacji dotyczącej wskazanego elementu protokołu. Znacznik ten zawiera atrybut „lang", którego wartością powinien być kod języka (kraju) - dwuliterowy, opisujący język w którym został przeprowadzony opis. Zawartością znacznika może być dowolny tekstem także wzbogacony o znaczniki języka HTML.

Schemat (XML Schema) deskryptora protokołu

```
<?xml version="1.0" encoding="iso8859-2" ?>
<xsd:schema xmlns:pg="http://loxim.mimuw.edu.pl/protoGen" xmlns:xsd
    ="http://www.w3.org/2001/XMLSchema" targetNamespace="http://
    loxim.mimuw.edu.pl/protoGen" elementFormDefault="qualified"
    attributeFormDefault="unqualified">

    <!-- ========================= SIMPLE TYPES
        ================================= -->

    <xsd:simpleType name="type">
        <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="uint8" />
                        <xsd:enumeration value="uint16" />
                        <xsd:enumeration value="uint32" />
                        <xsd:enumeration value="uint64" />
                        <xsd:enumeration value="sint8" />
                        <xsd:enumeration value="sint16" />
                        <xsd:enumeration value="sint32" />
                        <xsd:enumeration value="sint64" />
                        <xsd:enumeration value="varuint" />
                        <xsd:enumeration value="bool" />
                        <xsd:enumeration value="double" />
                        <xsd:enumeration value="bytes" />

                        <xsd:enumeration value="sstring" />
                        <xsd:enumeration value="string" />
                        <xsd:enumeration value="enum" />
                        <xsd:enumeration value="enum-map" />

                        <xsd:enumeration value="package" />
                        <xsd:enumeration value="package-map" />
        </xsd:restriction>
    </xsd:simpleType>
```

```
<!-- ================== COMPLEX TYPES
    ================================== -->
<!-- packets-->
<xsd:complexType name="descriptionType" mixed="true" >
    <xsd:sequence minOccurs="0" maxOccurs="1">
        <xsd:any processContents="lax"/>
    </xsd:sequence>
    <xsd:attribute name="lang" type="xsd:language"/>
</xsd:complexType>

<xsd:complexType name="fieldType">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="description" type="
            pg:descriptionType"/>
    </xsd:choice>
    <xsd:attribute name="name" type="xsd:NCName"/>
    <xsd:attribute name="type" type="pg:type"/>
    <xsd:attribute name="value" type="xsd:string"/>
    <xsd:attribute name="object-ref" type="xsd:NCName"/>
    <xsd:attribute name="object-ref-id" type="xsd:string"/>
    <xsd:attribute name="size" type="xsd:integer"/>
    <xsd:attribute name="nullable" type="xsd:boolean" use="
        optional" default="false"/>
 </xsd:complexType>

<xsd:complexType name="packetType">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="field" type="pg:fieldType"/>
        <!--xsd:element name="data-set-field" type="
            pg:dataSetFieldType"/-->
    </xsd:choice>
    <xsd:attribute name="name" type="xsd:NCName"/>
    <xsd:attribute name="extends-group"  type="xsd:NCName" use=
        "optional" />
    <xsd:attribute name="extends-packet" type="xsd:NCName" use=
        "optional" />
    <xsd:attribute name="id-value" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="packetsType">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="packet" type="pg:packetType"/>
    </xsd:choice>
    <!--xsd:attribute name="id-type" type="pg:type"/-->
</xsd:complexType>
<!-- enums -->

<xsd:complexType name="itemType">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
```

```
        <xsd:element name="description" type="
            pg:descriptionType"/>
    </xsd:choice>
    <xsd:attribute name="name" type="xsd:NCName"/>
    <xsd:attribute name="value" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="enumType">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="item" type="pg:itemType"/>
    </xsd:choice>
    <xsd:attribute name="name" type="xsd:NCName"/>
    <xsd:attribute name="as-type" type="pg:type"/>
</xsd:complexType>

<xsd:complexType name="enumsType">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="enum" type="pg:enumType"/>
    </xsd:choice>
</xsd:complexType>

<!-- metadata -->

<xsd:complexType name="languagesType">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="lang" type="pg:languageType"/>
  </xsd:choice>
</xsd:complexType>

<xsd:complexType name="languageType">
  <xsd:sequence minOccurs="0" maxOccurs="unbounded">
    <xsd:any processContents="skip"/>
  </xsd:sequence>
  <xsd:attribute name="id" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="metadataType">
    <xsd:all>
        <xsd:element name="major-version" type="xsd:integer"/>
        <xsd:element name="minor-version" type="xsd:integer"/>
        <!--xsd:element name="id-type" type="pg:type"/-->
        <xsd:element name="languages" type="pg:languagesType"
            minOccurs="0" maxOccurs="1"/>
    </xsd:all>
    <!--xsd:element name="size-header"-->
</xsd:complexType>

<xsd:complexType name="packetGroupType">
    <xsd:all minOccurs="1" maxOccurs="1">
```

```
            <xsd:element name="packets" type="pg:packetsType"/>
        </xsd:all>
        <xsd:attribute name="name" type="xsd:NCName"/>
        <xsd:attribute name="id-type" type="pg:type" use="optional"
            default="uint8"/>
        <xsd:attribute name="extends" type="xsd:NCName"/>
    </xsd:complexType>

    <xsd:complexType name="packetGroupsType">
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element name="packet-group" type="
                pg:packetGroupType"/>
        </xsd:choice>
    </xsd:complexType>

    <!-- protocol -->

    <xsd:complexType name="protocolType">
        <xsd:all>
          <xsd:element name="metadata" type="pg:metadataType" />
          <xsd:element name="enums" type="pg:enumsType" />
          <xsd:element name="packet-groups" type="
              pg:packetGroupsType" />
          <!-- xsd:element name="data-set-groups" type="
              pg:dataSetGroupsType"/-->
        </xsd:all>
    </xsd:complexType>

    <!-- ================== ROOT ========================== -->

    <xsd:element name="protocol" type="pg:protocolType"></
        xsd:element>
</xsd:schema>
```

D.2.3. Proste typy danych

Postanowienia ogólne

- Jeśli w sposób szczególny nie zaznaczono inaczej (a raczej nigdzie nie zaznaczono) wszystkie wartości zapisywane są w formacie Big-endian. W szczególności obejmuje to typy całkowitoliczbowe, rzeczywiste, oraz napisy w kodowaniu UTF-8 także stosują kolejność Big-endian.

Całkowitoliczbowe: uint8, sint8, uint16, sint16, int32, uint32, uint64, sint64

Pierwsza litera determinuje, czy mamy do czynienia z typem ze znakiem (u - unsigned), czy z typem bez znaku (s - signed). Liczba na końcu wyraża długość typu wyrażoną w bitach.

Typy są kodowane oczywiście w kolejności Big-endian.

varuint - Całkowitoliczbowy z kompresją (1,3,5 lub 9 bajtów)

Będzie to typ używany głównie do oznaczania długości stringów i paczek.

Rozwiązanie techniczne zostało zaczerpnięte z protokołu serwera MySQL [MYSQL].

Idea jest taka, że krótkie stringi (< 250znaków) będą się pojawiały najczęściej i chcemy mieć najmniejszy narzut na zapisanie długości (jedno bajtowy).

Zatem semantyka pierwszego bajtu jest następująca (w zależności od jego wartości)

0-249 Wartość ta jest jednocześnie wartością wynikową

250 Wartość jest null'em (zostawiamy dla umożliwienia stosowanie tego rozwiązania z systemami relacyjnymi)

251 Wartość ta poprzedza 2-bajtowe (uint16) pole w wartością właściwą

252 Wartość ta poprzedza 4-bajtowe (uint32) pole z wartością właściwą

253 Wartość ta poprzedza 8-bajtowe (uint64) pole z wartością właściwą. Nie należy jednak korzystać z wartości większych niż $2^{63} - 1$ (MAX_SINT64), ze względu na to, że w Javie nie są obsługiwane 8 bajtowe typy bez znaku.

string - Łańcuchy tekstu

Służy do przesyłania tekstów. Teksty te muszą być kodowane za pomocą UTF-8.

Format wartości typu string jest następujący: Najpierw idzie pole typu varuint - opisujące długość w bajtach następującego potem ciągu w UTF-8 (Big-endian).

sstring - Krótki łańcuch tekstu

Jest to tak naprawdę wariant typu string, ale ograniczony do łańcuchów nie dłuższych niż 249 bajtów. Czyli wtedy pole oznaczające długość ma jeden bajt. Jego wewnętrzna reprezentacja zupełnie się nie różni od typu string - został on wyróżniony formalnie - ze względu na uproszczenie notacji używanej przy opisach formatów poszczególnych paczek, a także ze względów bezpieczeństwa - gdyż pozwala zmniejszyć ryzyko typu DOS poprzez zawężenie wielkości danych, które mogą być zinterpretowane przez serwer.

bytes - Dane binarne

Służy do przesyłania danych binarnych (tablic bajtów).

Format wartości typu bytes jest następujący: Najpierw jest wysyłane pole typu varuint - opisujące długość w bajtach następującego potem ciągu bajtów.

D.3. Dokumentacja wygenerowanego kodu

D.3.1. Wygenerowany kod dla języka Java

Generator protokołów generuje kod w języku Java w wersji nie starszej niż Java 1.5, ponieważ korzysta z typów wyliczeniowych (enums) i szablonów (generics).

Zakładając, że w metadanych opisu protokołu dla języka java zadeklarowano packageName jako 'taki.sobie.protokol' (patrz: D.2.2) to zostanie wygenerowany następujący projekt:

pom.xml

Pom.xml (Project Object Model) jest to plik opisujący jak kompilować ten moduł za pomocą narzędzia „Maven2". Wskazuje on w szczególności biblioteki od których projekt zależy.

Moduł protokołu wygenerowany przez ProtoGen zależy od biblioteki: pl.edu.mimuw.loxim.protogen.lang.java.protolib:java_protolib, która zawiera definicję odpowiednich strumieni i klas bazowych (patrz: D.3.1).

Aby zbudować moduł wystarczy (zakładając, że użytkownik posiada zainstalowanego Maven'a 2 [1]) wykonać polecenie „mvn package" w katalogu zawierającym plik pom.xml. Wyniki procesu budowania zostaną umieszczone w katalogu target.

Typy wyliczeniowe - enumy

Każdy „enum" (typ wyliczeniowy) zostanie zapisany do katalogu ./src/main/java/taki/sobie/protokol/enums jako „Enum" języka Java (wersja 1.5+).

Typ „enum-map" jest odwzorcowywany na odpowiedni „java.util.EnumSet<typ bazowy>" (patrz: D.2.2).

Grupy paczek

Paczki grupy głównej zostaną zapisane w folderze ./src/main/java/taki/sobie/protokol/packages, a paczki z pozostałych grup w folderze ./src/main/java/taki/sobie/protokol/packages_[nazwa_grupy].

W każdym takim folderze powstanie także klasa będąca fabryką paczek, która udostępnia metodę:

public Package createPackage(long package_id) throws ProtocolException

Metoda ta tworzy paczkę w zależności od danego identyfikatora paczki.

Paczki

Dla każdej paczki zadeklarowanej w opisie protokołu jest tworzona odpowiednia klasa paczki. Wszystkie paczki rozszerzają następującą klasę:

```
public abstract class Package {

  /**
   * Metoda wypełnia pola paczki w oparciu o zserializowane dane
     zawarte w tablicy danej parametrem b.
```

[1] http://maven.apache.org/download.html

```
     * Tablica "b" powinna zawierać pełny opis paczki - włącznie z
        jej nagłówkiem.
     * */
    public void deserialize(byte[] b) throws ProtocolException;

    /**
     * Metoda sprawdza, czy dana paczka jest równa paczce na której
        wywołujemy tę metodę.
     * Za równe uznajemy te paczki, które są tego samego typu i
        których pola są równe co do zawartości.
     *
     * Metoda powinna być nadpisana przez deklarację każdej paczki.
     */
    public boolean equals(Object obj) {};

    /**
     * Metoda zapisuje do danego obiektu piszącego do strumienia
        wszystkie pola paczki (bez nagłówka).
     */
    public void deserializeContent(PackageInputStreamReader reader);

    /**
     * Metoda wczytuje paczkę z danego obiektu czytającego strumień.
     * Wczytywane są wszystkie pola paczki (bez nagłówka).
     */
    public void serializeContent(PackageOutputStreamWriter writer);

    /**
     * Metoda zapisuje do danego obiektu piszącego do strumienia
        wszystkie pola paczki (bez nagłówka).
     *
     * Metoda powinna być nadpisana przez deklarację każdej paczki.
     */
    protected abstract void deserializeW(PackageInputStreamReader
        reader)
        throws ProtocolException;

    /**
     * Metoda wczytuje paczkę z danego obiektu czytającego strumień.
     * Wczytywane są wszystkie pola paczki (bez nagłówka).
     *
     * Metoda powinna być nadpisana przez deklarację każdej paczki.
     */
    protected abstract void serializeW(PackageOutputStreamWriter
        writer)
        throws ProtocolException;
}
```

Ponadto każda paczka zawiera następujące elementy:

- Publiczną stałą ID zawierającą identyfikator paczki
- Bezparametrowy konstruktor paczki
- Konstruktor paczki, który przyjmuje wszystkie pola paczki jako swoje parametry
- Gettery i settery dla wszystkich pól paczki

Testy

Generator wygeneruje także testy:

./src/test/java/taki/sobie/protokol/tests/TestRunnerRec.java -
Klasa języka java dająca się uruchomić (zawiera metodę main), która uruchamia proces, który otwiera gniazdo TCPIP na wskazanym porcie i sprawdza, czy otrzymane paczki są zgodne z oczekiwanymi. Jedynym parametrem jaki można przekazać uruchamiając tę klasę jest numer portu na którym chcemy otworzyć gniazdo serwera.

Program ten można wykorzystywać do testów komunikacji z implementacjami protokołu dla innych języków programowania.

./src/test/java/taki/sobie/protokol/tests/TestRunnerSender.java -
Klasa języka java dająca się uruchomić (zawiera metodę main), która uruchamia proces, który wysyła na wskazany port na wskazanym serwerze paczki testowe. Uruchamiając musimy przekazać dwa parametry: nazwę hosta docelowego (lub adres IP), a także port do którego chcemy się podłączyć.

Program ten można wykorzystywać do testów komunikacji z implementacjami protokołu dla innych języków programowania.

./src/test/java/taki/sobie/protokol/tests/TestPackagesFactory.java -
Zawiera konstruktor wygenerowanych paczek testowych. Stanowi bazę dla innych testów.

./src/test/java/taki/sobie/protokol/tests/TestPackages.java -
Jest JUnitowym testem, który zapisuje wszystkie paczki testowe do strumienia, a następnie je wszystkie wczytuje i sprawdza, czy paczki są równe. Ten test jest używany w fazie testów w procesie budowania modułu przez Maven 2.

java_protolib

Biblioteka java_protolib.jar jest plikiem, który zawiera niezależną od konkretnego protokołu definicję podstawowych klas i interfejsów.

Poniżej listujemy kluczowe klasy znajdujące się w tej bibliotece:

pl.edu.mimuw.loxim.protogen.lang.java.template.layers.ProtocolLayer0
Podstawowa klasa używana w aplikacji do obsługi protokołu. Zawiera metody wysyłające przekazaną paczkę, a także czekającą i odczytującą paczkę. Klasa ponadto automatycznie odpowiada na paczkę „Ping" paczką „Pong".

pl.edu.mimuw.loxim.protogen.lang.java.template.exception.ProtocolException
Wyjątek rzucany gdy coś w protokole pójdzie nie tak jak powinno.

pl.edu.mimuw.loxim.protogen.lang.java.template.pstreams.PackageOutputStream

Strumień zakładany na OutputStream'ie, który udostępnia metodę writePackage(Package p).

pl.edu.mimuw.loxim.protogen.lang.java.template.pstreams.PackageInputStream

Strumień zakładany na InputStreamie, który udostępnia metodę readPackage().

pl.edu.mimuw.loxim.protogen.lang.java.template.streams.PackageOutputStreamWriter

Strumień zakładany na OutputStream'ie, który udostępnia metody pozwalające zapisać proste typy danych (patrz D.2.3).

pl.edu.mimuw.loxim.protogen.lang.java.template.streams.PackageInputStreamReader

Strumień zakładany na InputStream'ie, który udostępnia metody pozwalające odczytać proste typy danych (patrz D.2.3).

pl.edu.mimuw.loxim.protogen.lang.java.template.auth.AuthPassMySQL
Klasa pomocnicza oferująca wsparcie dla przeprowadzania autoryzacji taką metodą jak robi to serwer bazy danych MySQL („Password Algorithm" [**?**])

pl.edu.mimuw.loxim.protogen.lang.java.template.pstreams.PackagesFactory
Interfejs, który implementują wszystkie fabryki pakietów (patrz: D.3.1)

pl.edu.mimuw.loxim.protogen.lang.java.template.ptools.Package
Klasa bazowa dla wszystkich paczek.

D.3.2. Wygenerowany kod dla języka C++

Kod dla języka C++ jest samodzielny (jedyną zależnością jest OpenSSL potrzeby do działającej autoryzacji metodą zastosowaną w serwerze MySQL). Buduje się go za pomocą polecenia make wydanego w katalogu „protocol".

./make.defs

Zawiera definicję narzędzi i bibliotek używanych przez make.

Autentykacja - ./protocol/auth

W tym folderze zawarte są klasy pomocnicze potrzebne do przeprowadzenia autentykacji. Obecnie zaimplementowana jest tylko klasa AuthPassMySQL wspierająca autentykacją metodą zastosowaną w serwerze MySQL i wykorzystująca sumy kontrolne SHA1.

Strumienie paczek - ./protocol/pstreams

PackageOutputStream - Strumień zakładany na OutputStream;ie, który udostępnia metodę writePackage(Package* p).

PackageInputStream - Strumień zakładany na InputStream'ie, który udostępnia metodę readPackage() wczytującą dany pakiet ze strumienia.

Typy wyliczeniowe - ./protocol/enums

Każdy typ wyliczeniowy definiuje odpowiedni „enum" języka C++. Tworzony jest także typ mapy (zawierający pole bitowe dla każdej wartości typu wyliczeniowego).

Ponadto tworzona jest także fabryka umożliwiająca stworzenie instancji enuma lub mapy enumów na podstawie przekazanej wartości całkowitej, a także zapisanie enuma lub mapy enumów jako liczbę.

Grupy pakietów

Paczki grupy głównej zostaną zapisane w katalogu: ./protocol/packages, a grup nazwanych w ./protocol/packages_[nazwa_grupy].

Dla każdej grupy zostanie wygenerowana klasa, będąca fabryką paczek w zależności od przekazanego identyfikatora paczki.

Paczki

Dla każdej paczki zostaje zaimplementowana klasa analogiczna do tej opisanej dla języka Java (patrz: D.3.1).

Testy - ./protocol/tests

./protocol/tests/TestRunnerRec Uruchamia proces, który otwiera gniazdo TCPIP na wskazanym porcie i sprawdza, czy otrzymane pakiety są zgodne z oczekiwanymi. Jedynym parametrem jaki można przekazać uruchamiając tę klasę jest numer portu na którym chcemy otworzyć gniazdo serwera.

Program ten można wykorzystywać do testów komunikacji z implementacjami protokołu w innych językach programowania.

./protocol/tests/TestRunnerSender Uruchamia proces, który wysyła na wskazany port na wskazanym serwerze pakiety testowe. Uruchamiając musimy przekazać dwa parametry: nazwę hosta docelowego (lub adres IP), a także port do którego chcemy się podłączyć.

Program ten można wykorzystywać do testów komunikacji z implementacjami protokołu w innych językach programowania.

./protocol/tests/TestPackagesFactory Jest to fabryka wygenerowanych pakietów testowych. Stanowi bazę dla innych testów.

Gniazda - ./protocol/sockets

Katalog zawiera prostą, obiektową implementację gniazd. Jest ona wzorowana na tej dostępnej w języku Java i umożliwia łatwe używanie gniazd sieciowych (zarówno serwerowych jak i klienckich) i otwieranie strumieni na nich.

- ./protocol/sockets/TCPIPServerSocket
- ./protocol/sockets/TCPIPServerSingleSocket
- ./protocol/sockets/AbstractSocket
- ./protocol/sockets/TCPIPClientSocket

Strumienie ./protocol/streams

Katalog zawiera elementarne, obiektowe implementacje strumieni - wzorowane na języku Java.

./protocol/streams/AbstractOutputStream,AbstractInputStream - abstrakcyjne klasy bazowe dla strumieni

./protocol/streams/DescriptorInputStream,DescriptorOutputStream - prosta implementacja strumieni oparta o zapis i odczyt ze wskazanego deskryptora systemu operacyjnego (pliku, urządzenia, gniazda sieciowego, łącza).

./protocol/streams/PriorityOutputStream Klasa okalająca (wrapper) strumienia wyjściowego, który udostępnia dodatkową metodę writePriority(char*), która przy wielu odwołaniach do tego obiektu (z wielu wątków) zostanie wykonana przed wykonaniami zapisu w zwykłych metodach write(char*) (odpowiednia synchronizacja na semaforach).

Warstwy protokołu - ./protocol/layers/ProtocolLayer0

Podstawowa klasa używana w aplikacji do obsługi protokołu. Zawiera metody wysyłające przekazaną paczkę, a także metodę czekającą na i odczytującą paczkę. Klasa ponadto automatycznie odpowiada na paczki „Ping" paczkami „Pong".

Klasy pomocnicze

./protocol/ptools/Endianness -
Klasa zawiera metody ułatwiające konwersję Little/Big Endian dla typów prostych

./protocol/ptools/StringBufferStack -
Klasa umożliwia szybką konstrukcje napisów (buforów znaków) poprzez sklejanie napisów (tablic znaków).

./protocol/ptools/Package -
Klasa bazowa dla wszystkich paczek

./protocol/ptools/CharArray -
Tablica znaków z określonym rozmiarem

./protocol/ptools/PackageBufferWriter -
Klasa zawierająca metody zapisujące do danego strumienia typy proste.

./protocol/ptools/PackageBufferReader -
Klasa zawierająca metody odczytujące z danego strumienia typy proste.

Przykładowy kod używający protokół po stronie serwera

```
#include <stdio.h>
#include <stdlib.h>

#include "./ptools/Package.h"
#include "./sockets/TCPIPServerSocket.h"
#include "./layers/ProtocolLayer0.h"
```

```
#include "./packages/W_c_helloPackage.h"
#include "./packages/A_sc_byePackage.h"

using namespace protocol;

int main(int argc,char** argv)
{
  printf("Opening server socket on port: %d\n",atoi(argv[1]));

  TCPIPServerSocket* serversocket=new TCPIPServerSocket(NULL,atoi(
     argv[1]));
  serversocket->bind();
  AbstractSocket *socket=serversocket->accept();

  ProtocolLayer0 *pl0=new ProtocolLayer0(socket);

  Package* p=pl0->readPackage();
  if((p)&&(p->getPackageType()==ID_W_c_helloPackage))
  {
    printf("Received package\n");
    delete p;

    printf("Sending response...\n");
    p=new A_sc_byePackage();
    pl0->writePackage(p);
    delete p;
  }else{
    printf("Unexpected package\n");
    if (p)
      delete p;
  }

  delete pl0;
  socket->close();
  delete socket;
  serversocket->close();
  delete serversocket;
  printf("Finished");
}
```

Przykładowy kod używający protokół po stronie klienta

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

#include "sockets/TCPIPClientSocket.h"
#include "layers/ProtocolLayer0.h"
```

```
#include "./packages/W_c_helloPackage.h"
#include "./packages/A_sc_byePackage.h"
#include "./enums/CollationsFactory.h"

using namespace protocol;

int main(int argc, char** argv) {
  printf("Opening server socket on: %s:%d\n", argv[1], atoi(argv
     [2]));

  TCPIPClientSocket* clientsocket=new TCPIPClientSocket(argv[1],
     atoi(argv[2]));
  clientsocket->connect();

  ProtocolLayer0 *pl0=new ProtocolLayer0(clientsocket);

  printf("Sending invitation...\n");
  Package * p=new W_c_helloPackage(getpid(),new CharArray("
     testClient"),
      NULL,NULL,NULL, coll_unicode_collation_alghorithm_uts10,2);
  pl0->writePackage(p);
  delete pl0;

  printf("Waiting for package...");
  p=pl0->readPackage();
  printf("... got it");
  if ((p)&&(p->getPackageType()==ID_A_sc_byePackage)) {
    printf("Received package\n");
    delete p;
  } else {
    printf("Unexpected package\n");
    if (p)
      delete p;
  }

  delete pl0;
  clientsocket->close();
  delete clientsocket;
  printf("Finished");
}
```

D.4. Dokumentacja programistyczna

D.4.1. Architektura rozwiązania

Java

ProtoGen został napisany w języku Java co umożliwia używanie go na większości platform sprzętowych i z wykorzystaniem wielu systemów operacyjnych.

Plexus - zorientowana na komponenty

ProtoGen został napisany w architekturze zorientowanej na komponenty. Do zarządzania komponentami używamy biblioteki Plexus (http://plexus.codehaus.org). Z punktu widzenia użytkownika i programisty, zastosowanie tej architektury umożliwia stosunkowo łatwe dodanie obsługi nowego języka generowanego kodu. Wymaga ono jedynie stworzenia implementacji interfejsu: „pl.edu.mimuw.loxim.protogen.api.PartialLanguageGenerator", a następnie zadeklarowanie jej w lokalnym deskryptorze components.xml jar'a zawierającego tę implementację jako:

```
<?xml version="1.0" encoding="iso8859-2"?>
<component-set>
    <components>
        <component>
            <role>pl.edu.mimuw.loxim.protogen.api.
                PartialLanguageGenerator</role>
            <role-hint>NowyJęzyk</role-hint>
            <implementation>pl.edu.mimuw.loxim.protogen.lang.nl.
                NewLanguageGenerator</implementation>
        </component>
        <component>
            <role>pl.edu.mimuw.loxim.protogen.api.LanguageGenerator
                </role>
            <role-hint>NowyJęzyk</role-hint>
            <implementation>pl.edu.mimuw.loxim.protogen.api.
                BasicLanguageGenerator</implementation>
            <requirements>
                <requirement>
                    <role>pl.edu.mimuw.loxim.protogen.api.
                        PartialLanguageGenerator</role>
                    <role-hint>NowyJęzyk</role-hint>
                </requirement>
            </requirements>
        </component>
    </components>
</component-set>
```

W momencie, gdy taki jar znajdzie się w classpath'ie uruchamianego ProtoGen'a, to język „NowyJęzyk" będzie dostępny jako jeden z docelowych język dla generowanego kodu.

Reasumując, główną korzyścią zastosowania architektury zorientowanej na komponenty, jest umożliwienie zewnętrznym dostawcom dodawanie implementacji nowych języków bez żadnej ingerencji w dostępny dotychczas kod.

D.4.2. Proces budowania aplikacji

1. Zainstalować Maven2 (http://maven.apache.org/download.html)

2. Wejść do katalogu głównego ProtoGen'a i wydać polecenie „mvn clean package assembly:assembly"

3. W podkatalogu „target/protogen-x.xx.xx-bin.dir" będzie się znajdowała wersja dystrybucyjna projektu.

D.4.3. Dalszy rozwój

Złożona logika

Obecnie ProtoGen nie wspiera złożonej logiki w konstrukcji pakietów. Niemożliwe jest warunkowanie istnienia pola w zależności od wartości innego pola, a także tworzenia tablicy pól. Obecnie ProtoGen przewiduje możliwość nadpisania wygenerowanego kodu. W ten sposób można - nadpisując kod obsługi odpowiedniej paczki - uzyskać obsługę dowolnie złożonej logiki.

Żeby zminimalizować konieczność wykorzystania takiego obejścia, a tym samym uzyskać „pełność" rozwiązania przewidujemy w przyszłych wersjach możliwość wprowadzenia następujących atrybutów do pola field (patrz: D.2.2):

if-field — Zawiera nazwę pola z bieżącej paczki (wcześniej zadeklarowanego), którego wartość będziemy porównywali do „if-value".

if-value — Jeżeli zawartość pola o nazwie zadanej atrybutem „if-field" jest równa wartości „if-value" to opisywane pole występuje w tej paczce. W przeciwnym przypadku nie występuje.

repeat — Zawiera nazwę pola całkowitoliczbowego, które zawiera informację ile razy bieżące pole jest powtórzone.

Więcej języków programowania

System jest przystosowany do rozszerzania o generatory do innych języków programowania bez modyfikacji kodu źródłowego projektu bazowego. Dodanie obsługi nowego języka programowania wymaga napisania komponentu systemu Plexus implementującego interfejs „pl.edu.mimuw.loxim.protogen.api.PartialLanguageGenerator".

Oczywistym jest, że dodanie generatorów do języków programowania takich jak, C#, Python, Ruby, PHP, Ocaml istotnie zwiększy użyteczność tego projektu.

Generowanie dokumentacji

Podobnie jak z dodatkowymi językami programowania, wydaje się istotną z punktu widzenia użytkownika możliwość uzyskania aktualnej dokumentacji protokołu. Cechą pożądaną jest by ProtoGen był wstanie wygenerować dokumentację - najlepiej do wielu najpopularniejszych formatów (HTML, LaTeX, PDF, ODF).

By uniknąć ręcznej obsługi tych wszystkich formatów - wydaje się uzasadnionym użycie narzędzia typu Doxia (http://maven.apache.org/doxia), które umożliwia konwersje pomiędzy wieloma formatami „bogatego" tekstu.

Wsparcie dla wielu wersji pobocznych

Zarówno definicję paczki, jak i pola można rozszerzyć o atrybut „since-minor-version", który będzie specyfikował od której wersji pobocznej protokołu dane pole występuje.

Dodatkowo wymagamy, by kolejność wartości „since-minor-version" zgadzała się z kolejnością pól w paczce i by każde pole posiadające ten atrybut posiadało także zadeklarowaną wartość domyślną (pole „value"). Warunki te są wystarczające, by zachować zgodność protokołu pomiędzy różnymi wersjami pobocznymi.

Dodatek E

Zawartość płyty

Do pracy została załączona płyta CD-ROM z zawartością zorganizowaną w następujący sposób:

./dokumenty/0. Praca magisterska - Piotr Tabor.pdf

./dokumenty/1. Analiza starego protokołu dla LoXiM.pdf

./dokumenty/2. Dokumentacja nowego protokołu dla LoXiM.pdf

./dokumenty/3. Analiza możliwości wykorzystania protokołu LDAP dla SBQL DB.pdf

./dokumenty/4. Dokumentacja generator protokołów.pdf

./dokumenty/src/ - Folder zawiera źródłowe pliki tex, a także obrazki potrzebne do wygenerowanie w/w dokumentów przy pomocy systemu LaTeX

./protogen/ - Folder zawiera pliki generatora protokołów „ProtoGen"

./protogen/src/ - pliki źródłowe - do kompilacji za pomocą narzędzia „Maven2"

./protogen/src/api/ - API generatora protokołów z którego korzystają generatory dla poszczególnych języków

./protogen/src/core/ - rdzeń wykonywalne aplikacji „ProtoGen"

./protogen/src/langs/ - implementacje generacji kodu do różnych języków

./protogen/src/langs/cpp/ - do C++

/protogen/src/langs/java/ - do Javy

./protogen/src/langs/java_protolib/ - biblioteka, z której korzysta wygenerowany kod dla Javy

./protogen/bin/ - skompilowana wersja generatora protokołów „ProtoGen"

./protogen/bin/*.jar - biblioteki zawierające „ProtoGen" i jego zależności

./protogen/bin/protoGen.sh - skrypt uruchamiający „ProtoGen" w systemach Unixowych (bash)

./protogen/bin/protoGen.bat - skrypt uruchamiający „ProtoGen” w systemach firmy Microsoft

./protogen/example/ - katalog zawiera zestaw danych potrzebnych do wygenerowania protokołu

./protogen/example/over/ - dla LoXiMa przy użyciu załączonego generatora ”ProtoGen”

./protogen/example/over/cpp/ - zestaw plików do nadpisania przy generowaniu do C++

./protogen/example/over/java/ - zestaw plików do nadpisania przy generowaniu do Javy

./protogen/example/loxim2.xml - opis XML protokołu dla LoXiMa

./protogen/example/run.sh - skrypt, który uruchamia generowanie kodu do C++ i Javy

./loxim_protocol/ - katalog zawiera wygenerowane biblioteki na potrzeby systemu LoXiM:

./loxim_protocol/cpp/ - w języku C++

./loxim_protocol/java/ - w języku Java

./licencje - katalog zawiera treści licencji na których znajduje się sama praca,a także tych na których zostały wydane wykorzystywane w pracy komponenty i biblioteki.

Bibliografia

[OSI] OSI Reference Model ISO/IEC standard 7498-1:1994, ISO, [20.05.2008] (http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip)

[DoD] Architectural Principles of the Internet, RFC 1958, B. Carpenter, June 1996

[MYSQL] MySQL Protocol - version 10. Ian Redfern, [03.12.2006], (http://www.redferni.uklinux.net/mysql/MySQL-Protocol.html)

[PGSQL] PostgreSQL Frontend/Backend Protocol - version. PostgreSQL Fundation, [03.12.2006], (http://developer.postgresql.org/pgdocs/postgres/protocol.html)

[SBQL] Stack-Based Approach (SBA) and Stack-Based Query Language (SBQL). Kazimierz Subieta, [03.12.2006], (http://www.sbql.pl)

[TDS] TDS Protocol Documentation. FreeTDS, [10.12.2006], (http://www.freetds.org/tds.html)

[UTS10] Unicode Technical Standard #10 Unicode Collation Algorithm. Mark Davis, Ken Whistler, [29.05.2008], (http://www.unicode.org/reports/tr10/)

[ISO8601] ISO 8601 - Numeric representation of Dates and Time, ISO, [29.05.2008], (http://www.iso.org/iso/support/faqs/faqs_widely_used_standards/ widely_used_standards_other/date_and_time_format.htm)

[ASN1] Communication between heterogeneous systems, Olivier Dubuisson, 2008, (http://www.oss.com/asn1/bookreg2.html)

[LDAP] Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map, K. Zeilenga, Ed.[06.2006]

[RFC4511] Sermersheim, J., Ed., "Lightweight Directory Access Protocol (LDAP): The Protocol", RFC 4511, June 2006.

[RFC4512] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP): Directory Information Models", RFC 4512, June 2006.

[RFC4513] Harrison, R., Ed., "Lightweight Directory Access Protocol (LDAP): Authentication Methods and Security Mechanisms", RFC 4513, June 2006.

[RFC4514] Zeilenga, K., Ed., "Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names", RFC 4514, June 2006.

[RFC4515] Smith, M., Ed. and T. Howes, "Lightweight Directory Access Protocol (LDAP): String Representation of Search Filters", RFC 4515, June 2006.

[RFC4516] Smith, M., Ed. and T. Howes, "Lightweight Directory Access Protocol (LDAP): Uniform Resource Locator", RFC 4516, June 2006.

[RFC4517] Legg, S., Ed., "Lightweight Directory Access Protocol (LDAP): Syntaxes and Matching Rules", RFC 4517, June 2006.

[RFC4519] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP): Internationalized String Preparation", RFC 4518, June 2006.

[RFC4519] Sciberras, A., Ed., "Lightweight Directory Access Protocol (LDAP): Schema for User Applications", RFC 4519, June 2006.

[RFC5420] Zeilenga, K., "Internet Assigned Numbers Authority (IANA) Considerations for the Lightweight Directory Access Protocol (LDAP)", BCP 64, RFC 4520, June 2006.

[RFC4521] Zeilenga, K., "Considerations for LDAP Extensions", BCP 118, RFC 4521, June 2006.

[X.500] [X.500] International Telecommunication Union - Telecommunication Standardization Sector, "The Directory – Overview of concepts, models and services", X.500(1993) (also ISO/IEC 9594-1:1994).

[X.501] International Telecommunication Union - Telecommunication Standardization Sector, "The Directory – Models", X.501(1993) (also ISO/IEC 9594- 2:1994).

[X.511] International Telecommunication Union - Telecommunication Standardization Sector, "The Directory: Abstract Service Definition", X.511(1993) (also ISO/IEC 9594-3:1993).