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 A).

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 **??**

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 **??**

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 **??**

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 **??**

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 **??**).

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 **??**).

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 **??** i **??**.

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 **??**), 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 **??**) 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

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)