# Analiza możliwości wykorzystania protokołu LDAP do obsługi dostępu do semistrukturalnej BD

Piotr Tabor (pt214569@students.mimuw.edu.pl)

4 czerwca 2008

**Historia**

| Data | Wersja | Autor | Zmiany |
|---|---|---|---|
| 2008-03-09 – 2008-03-26 | 0.1 | Piotr Tabor | Pierwsza wersja dokumentu |
| 2008-03-09 – 2008-06-03 | 0.2 | Piotr Tabor | Analiza i podsumowanie |

## Spis treści

# 1 Wprowadzenie

## 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.

## 1.2 Usługi katalogowe

## 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".

## 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.

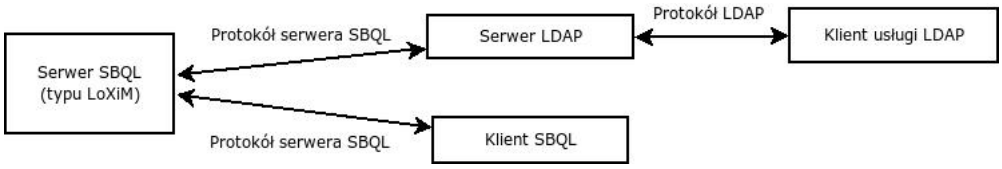
W związku z tym, że mechanizm zapytań LDAP jest stosunkowo prosty — umożliwia on bardzo efektywną realizację operacji (w pełni równoległe i z wykorzystaniem niewielkiej pamięci pomocniczej przetwarzanie). Ogromną wadę stanowi natomiast niewielka ilość operacji, którą z użyciem tego narzędzia można przeprowadzić.

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.

### 1.4.1 Serwer SBQL jako zaplecze (ang. backend) usługi LDAP




Rysunek 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:

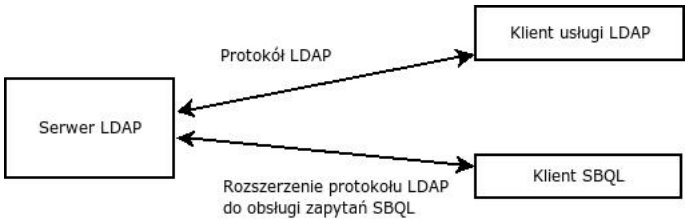
- Nie istnieje żadna zależność pomiędzy protokołem dostępu do bazy danych SBQL, a protokołem LDAP.

- 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 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.

### 1.4.2 Serwer LDAP świadczący dostęp za pomocą języka SBQL



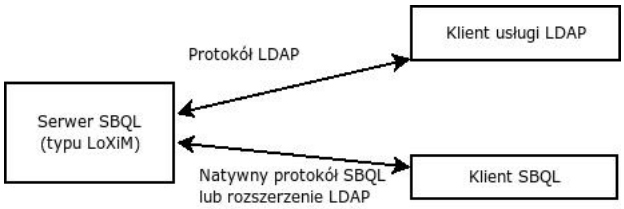

Rysunek 2: Schemat serwera LDAP z dostępem przez SBQL

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 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 2.4.1.

### 1.4.3 Serwer SBQL świadczący dostęp do danych za pomocą protokołu LDAP




Rysunek 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 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 2.4.1.

# 2 Analiza

## 2.1 Porównanie modeli danych

### 2.1.1 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" [**?**], 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.

### 2.1.2 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.

## 2.2 Symulowanie modelu danych protokołu LDAP za pomocą modelu danych SBQL

### 2.2.1 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):

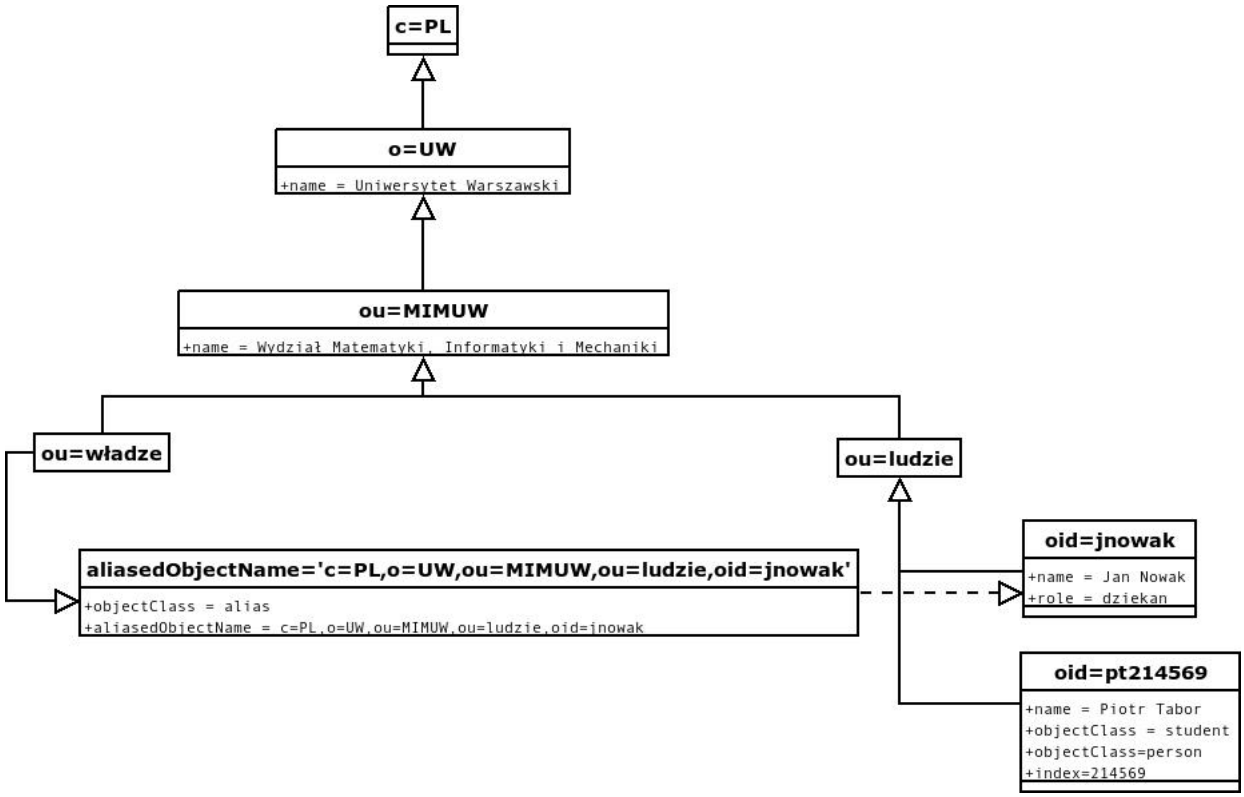
Tworzymy obiekt okalający o id ($i_1$), zawierające następujący wiązania (bindings):

**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#):




Rysunek 4: Przekład 1 danych w modelu LDAP

```
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=jnowak'
            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#

## 2.3 Operacje protokołu LDAP

### 2.3.1 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.

### 2.3.2 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 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.

### 2.3.3 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.

### 2.3.4 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 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).

### 2.3.5 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.

### 2.3.6 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.

### 2.3.7 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.

### 2.3.8 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.

## 2.4 Podsumowanie

Pomimo wielu cech wspólnych obu narzędzi (patrz 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 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).

### 2.4.1 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.

# Literatura


[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).