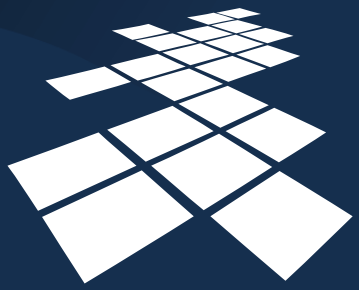
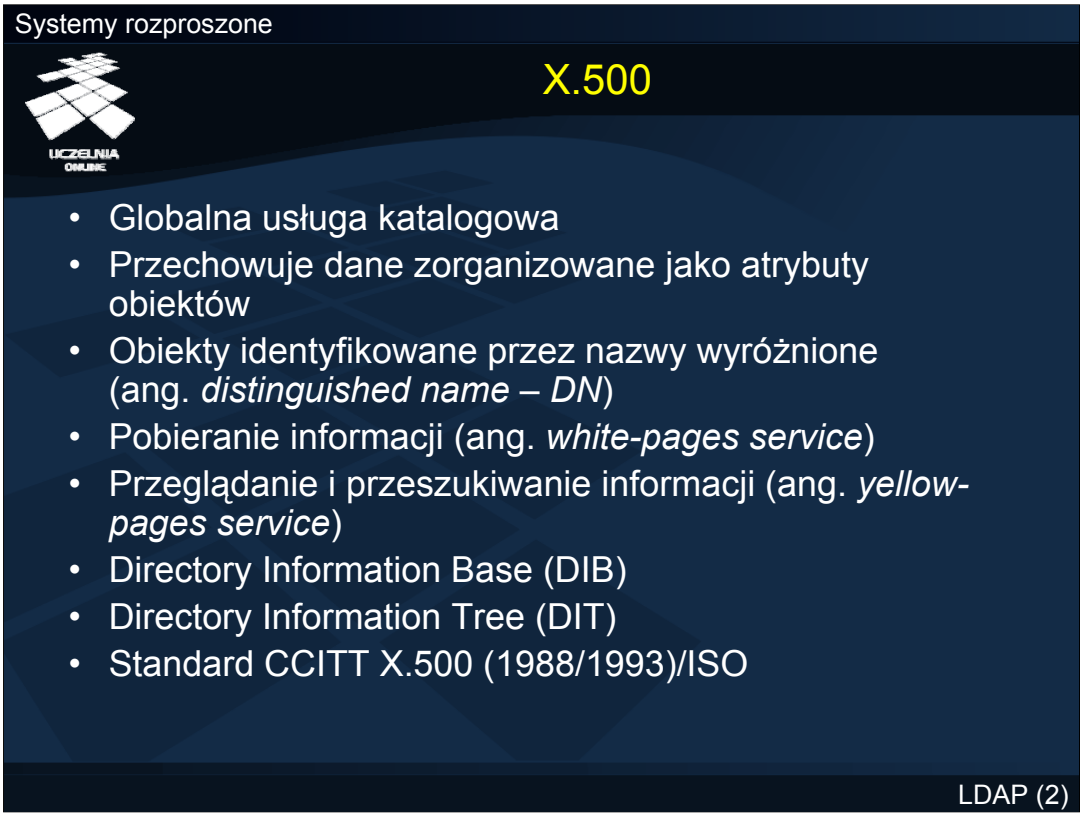
LDAP

Cezary Sobaniec



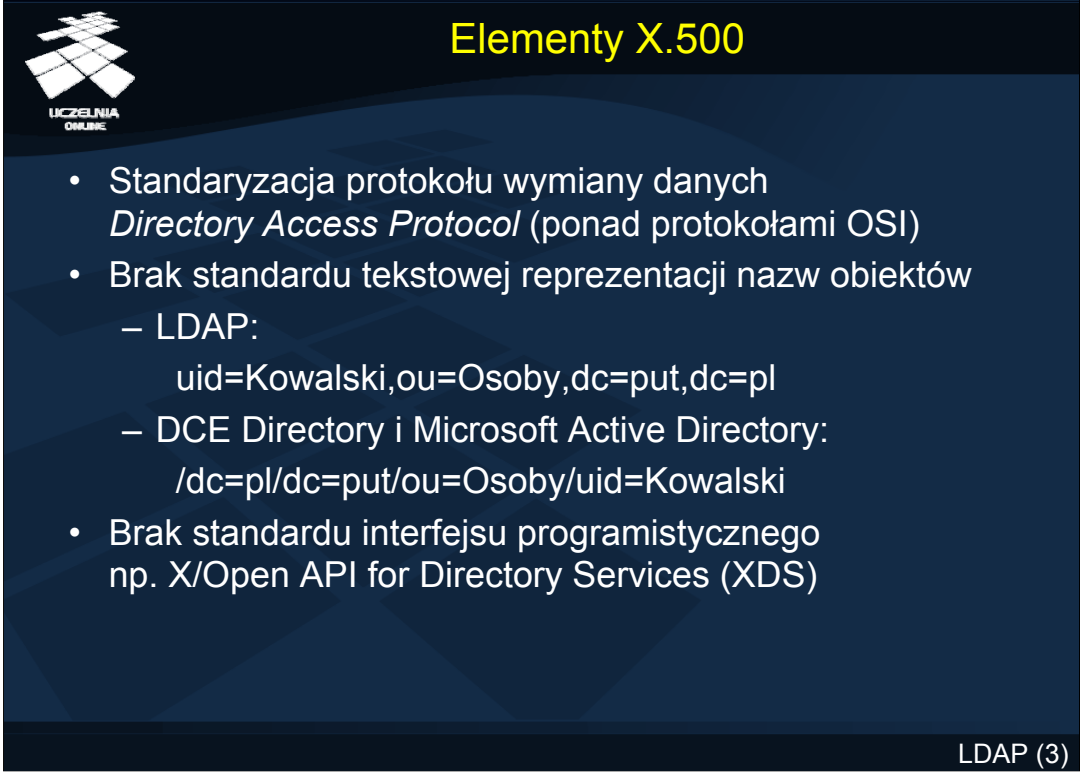
UCZELNIA
ONLINE



X.500 jest usługą katalogową. Usługa katalogowa (ang. *directory service*) jest usługą nazewniczą, dzięki której użytkownik może nie tylko uzyskać proste odwzorowanie nazwy na adres, ale również może dokonywać przeszukiwania bazy danych na podstawie *treści*. Rozróżnienie to można w prosty sposób odnieść do naszych codziennych doświadczeń, kiedy poszukujemy informacji. Zwykła książka telefoniczna reprezentuje klasyczne usługi nazewnicze; użytkownik zna nazwisko osoby, której numer telefonu poszukuje, i na tej podstawie uzyskuje odpowiednią informację. Tego typu usługi nazewnicze nazywane są też niekiedy *white-pages service* (chodzi o białe strony typowej książki telefonicznej). Inaczej sytuacja wygląda, gdy użytkownik potrzebuje uzyskać adres czy numer telefonu osoby wykonującej pewien zawód czy czynność. Szukając lekarza czy mechanika samochodowego sięgamy po żółte strony książki telefonicznej i poszukujemy informacji w odpowiednim miejscu *katalogu* profesji. Na tej zasadzie działają właśnie usługi typu *yellow-pages*, których przykładem jest usługa katalogowa X.500.

Usługa katalogowa X.500 jest usługą globalną, podobnie jak usługa DNS. Oznacza to, że lokalnie istniejące usługi mogą być w przyszłości zintegrowane i mogą tworzyć jedną, ogromną bazę skatalogowanej informacji. Baza ta określana jest mianem **bazy informacji katalogowej** (ang. *Directory Information Base – DIB*). Informacja zorganizowana jest w postaci **drzewa informacji katalogowej** (ang. *Directory Information Tree – DIT*). W drzewie tym przechowywane są obiekty, które przechowują informację w postaci atrybutów o określonych nazwach. Każdy obiekt w drzewie jest jednoznacznie identyfikowany poprzez **nazwę wyróżnioną** (ang. *distinguished name*), oznaczaną jako DN.

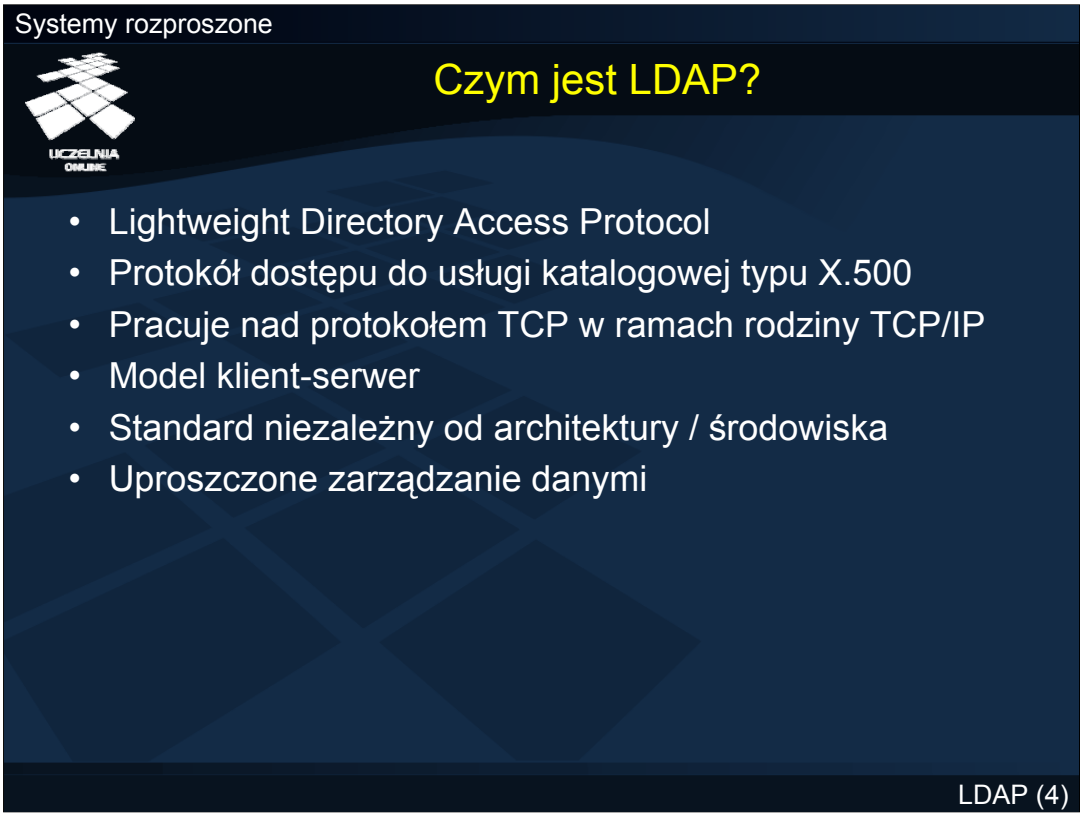
Usługa X.500 jest standardem ISO.



Standard X.500 zawiera opis usługi katalogowej, nie specyfikując jednak wielu elementów niezbędnych do implementacji tej usługi, pozostawiając decyzję dostawcom oprogramowania. Podstawowym elementem usługi, który jest objęty precyzyjną standaryzacją jest protokół komunikacyjny stosowany do wymiany danych klient-serwer i serwer-serwer. Jest to protokół *Directory Access Protocol* – DAP. Protokół ten został zdefiniowany dla rodziny protokołów OSI (ang. *Open Standard Interconnection*). Niestety protokoły OSI nie zdobyły dużej popularności, m.in. z racji ich dużego stopnia złożoności. Obecnie jak wiadomo zdecydowanie najbardziej popularną rodziną protokołów sieciowych jest rodzina TCP/IP.

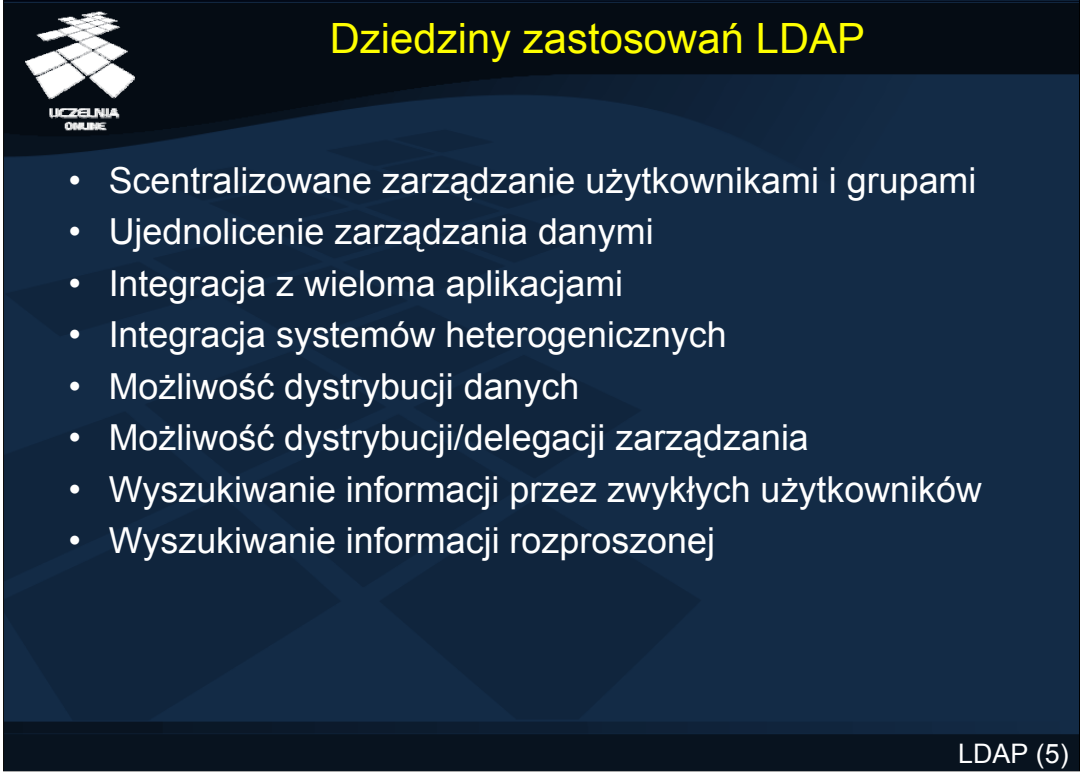
W usłudze X.500 nie ma sprecyzowanego standardu tekstowej reprezentacji nazw obiektów. Standard dotyczy jedynie binarnej reprezentacji na poziomie protokołu. Z tego powodu oprogramowanie pochodzące od różnych dostawców może stosować różne konwencje. Na slajdzie pokazano przykładowe dwie konwencje zapisu nazwy wyróżnionej obiektu stosowanej w dwóch różnych produktach.

Standard X.500 nie specyfikuje również standardu interfejsu programistycznego do programowania dostępu do usług katalogowych. Istnieje więc tyle interfejsów ilu dostawców oprogramowania.



Protokoły OSI są bardzo złożone, co było przyczyną trudności w ich szerokiej akceptacji i wdrożeniu. Podobnie sytuacja wyglądała z protokołem DAP – był tak rozbudowany, że jego efektywna implementacja na komputerach, które były dostępne 20 lat temu była niemal niemożliwa (zważywszy, że miała to być tylko usługa pomocnicza w systemie). To wszystko sprawiło, że protokół DAP funkcjonował głównie na papierze. Wraz z rozwojem technologii informatycznych potrzeba korzystania z usług katalogowych stawała się jednak coraz bardziej aktualna. Zaproponowano więc protokół, który będzie funkcjonował na bazie protokołu transportowego TCP, a więc w ramach *de facto* standardowej rodziny TCP/IP, i który nie będzie tak rozbudowany jak DAP. Protokół nazwano *Lightweight Directory Access Protocol*, czyli odchudzony DAP. Jest to protokół dostępu do usługi katalogowej typu X.500. W przeciwieństwie jednak do protokołu DAP, LDAP nie wymaga dużej ilości zasobów komputerowych.

Protokół LDAP zakłada wykorzystanie niezawodnego, połączeniowego protokołu transportowego. Jest to protokół dla usługi działającej w modelu klient-serwer. Protokół jest otwarty, ustandaryzowany, niezależny od architektury systemu komputerowego czy systemu operacyjnego. W protokole, dla zwiększenia efektywności, zakłada się uproszczone zarządzanie danymi.



Usługi katalogowe mogą pełnić rolę integracyjną dla systemów komputerowych. Umożliwiają przeprowadzenie centralizacji zarządzania informacją katalogową, a więc np. informacją o użytkownikach, grupach użytkowników czy zasobach wykorzystywanych w sieci. Ze względu na swoją neutralność platformową usługa może być wykorzystana do integracji systemów heterogenicznych. W ramach natomiast pojedynczych systemów może udostępniać swoje dane dla wielu aplikacji czy innych usług systemowych. Jest to bardzo pożądane z punktu widzenia administracyjnego, gdyż pozwala właśnie scentralizować i ujednolicić zarządzanie zasobami.

Dane przechowywane w usługach katalogowych, pomimo koncepcyjnej centralizacji, nie muszą być przechowywane i zarządzane przez pojedynczy system komputerowy. Podobnie jak w przypadku usługi DNS można bazę informacji katalogowej rozdystrybuować pomiędzy wiele serwerów w celu zwiększenia efektywności dostępu do danych i również samej dostępności. Podobnie wygląda kwestia dystrybucji odpowiedzialności za zarządzanie poszczególnymi fragmentami struktury katalogowej. Można dokonywać *delegowania* tej odpowiedzialności do jednostek podrzędnych, które będą samodzielnie mogły dokonywać odpowiedniej rekonfiguracji.

Dane przechowywane w usługach katalogowych mogą być udostępniane szerokiemu gronu użytkowników. Dzięki funkcjonalności protokołów dostępu do usług katalogowych jest możliwe wygodne przeszukiwanie zawartości bazy, nawet jeżeli baza ta jest fizycznie rozproszona. Informacje mogą być chronione i udostępniane tylko użytkownikom uprawnionym.

LDAP a bazy danych

- Zdecydowana przewaga odczytów: optymalizacja
- Rozszerzalność schematów danych
- Dystrybucja danych
- Zwielokrotnianie
- Przetwarzanie transakcyjne
  - uproszczony model: zmiana pojedynczego rekordu/obiektu
- Rozmiar danych
  - LDAP: małe porcje informacji
- Standaryzacja protokołu
  - dowolny klient LDAP — dowolny serwer LDAP

Baza informacji katalogowej przechowywana i zarządzana przez usługę katalogową może być traktowana jako po prostu baza danych. Powstaje więc uzasadnione pytanie dlaczego dla potrzeb przechowywania tych informacji nie stosuje się sprawdzonych i efektywnych rozwiązań opartych np. na relacyjnych bazach danych. Otóż istnieje kilka istotnych różnic odnośnie funkcjonowania tych systemów.

Usługi katalogowe w zdecydowanej większości są przeznaczone *odczytywania* danych. Zdecydowana przewaga odczytów przekłada się na algorytmy dostępu do danych, które starają się optymalizować przede wszystkim tego typu operacje. W bazach danych duży nacisk kładzie się na efektywne zarządzanie współbieżnym wykonywaniem transakcji, w których występują zapisy.

Relacyjne bazy danych mają statyczne schematy danych. Raz zaprojektowana aplikacja korzysta cały czas z tabel o tej samej strukturze. Zmiana struktury bazy danych wymaga w większości przypadków aktualizacji programu. W przypadku usług katalogowych schematy danych mogą być na bieżąco modyfikowane i w większości przypadków nie stanowi to problemu dla aplikacji, które się odwołują do usług katalogowych. Zmiana schematu danych może oznaczać tu, że obiekt uzyskuje nowe atrybuty, traci stare, lub dodawane są do niego nowe węzły podrzędne.

Dane przechowywane w usłudze katalogowej mogą być dość swobodnie dystrybuowane do różnych, rozproszonych serwerów. W przypadku baz danych dystrybucja niekiedy jest możliwa, ale nie jest to działanie standardowe i wymaga specjalnego potraktowania.

Serwery usług katalogowych z reguły oferują możliwość zwielokrotniania danych. Zadanie to jest stosunkowo proste do zrealizowania chociażby ze względu na zdecydowaną przewagę operacji odczytu, co redukuje prawdopodobieństwo pojawienia się problemów ze spójnością.

Nowoczesne bazy danych oferują możliwości przetwarzania transakcyjnego, gdzie zapewnione są własności atomowości, spójności, izolacji i trwałości. W przypadku usług katalogowych transakcje występują w postaci bardzo zredukowanej; modyfikacje obejmują pojedyncze obiekty.

Bazy danych operują na rekordach, które mogą mieć duże rozmiary (np. obiekty BLOB). W usługach katalogowych na poszczególne obiekty składają się ich atrybuty, które same z siebie raczej mają niewielkie rozmiary.

W bazach danych (względnej) standaryzacji podlega język zapytań SQL. Sam protokół dostępu do bazy danych jest natomiast zależny od producenta. W przypadku usług katalogowych to właśnie protokół jest elementem standardowym, co (teoretycznie) powinno zagwarantować możliwość połączenia klienta od jednego dostawcy z serwerem innego dostawcy.

LDAP a NIS (Network Information System)

Problemy z usługą NIS

- używa dowolnych numerów portów (RPC)
- brak mechanizmów szyfrowania komunikacji
- brak mechanizmów kontroli dostępu
- płaska przestrzeń nazw
  - problem skalowalności
  - NIS+
- dane indeksowane pojedynczym kluczem
  - ograniczone możliwości wyszukiwania informacji
- zmiany wprowadzane tylko przez administratora na centralnym serwerze

Usługa NIS (ang. *Network Information System*) realizuje cele, które w dużym zakresie pokrywają się z celami stosowania usług katalogowych. Usługa NIS ma jednak kilka istotnych wad wymienionych na slajdzie.

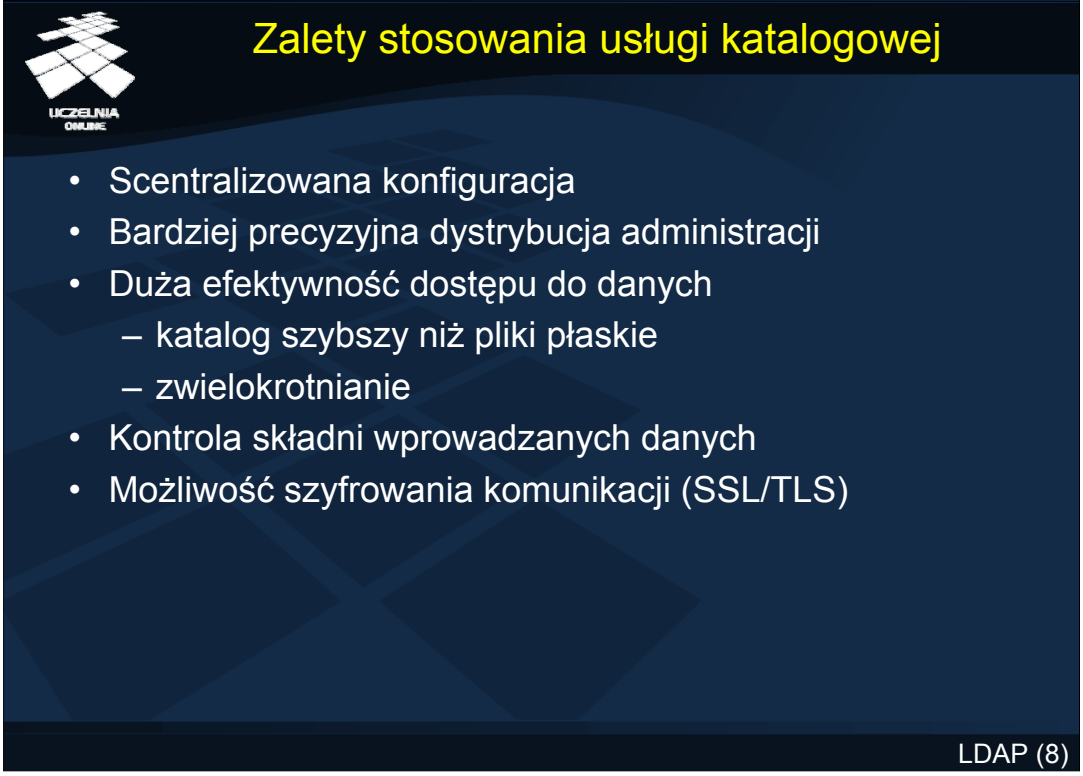
NIS został zrealizowany z wykorzystaniem mechanizmu zdalnych wywołań procedur RPC, które wykorzystują w warstwie transportowej różnych numerów portów, utrudniając lub wręcz uniemożliwiając bezpieczne wdrożenie usługi. Jest to tym trudniejsze, że komunikacja nie może być szyfrowana, co dodatkowo ogranicza stosowalność NIS-a.

System NIS nie oferuje drobnoziarnistego mechanizmu kontroli dostępu do danych. Jedynym sposobem ograniczenia dostępu jest wskazanie dopuszczalnych adresów IP dla klientów. W przypadku usługi katalogowej kontrola dostępu jest bardzo szczegółowa i można ją definiować w odniesieniu do poszczególnych użytkowników i obiektów z drzewa katalogowego.

System NIS oferuje jedynie płaską przestrzeń nazw. Możliwe jest przypisywanie obiektów do domen, ale domeny nie mogą tworzyć hierarchii. Rodzi to istotne problemy w przypadku dużych instalacji. Zawartości serwera nie może być rozdystrybuowana na kilka lokalizacji. Nie można więc również dzielić kompetencji administracyjnych na wiele ośrodków. Z problemami tymi radzi sobie częściowo system NIS+ wprowadzając hierarchiczne domeny. Niestety jest dużo bardziej złożony od strony administracyjnej i przez to mało popularny.

W systemie NIS dane są indeksowane zawsze pojedynczym kluczem. Oznacza to niemożliwość wyszukiwania informacji wg dowolnego atrybutu. W przypadku usług katalogowych całkowicie naturalne jest wyszukiwanie informacji po wartościach dowolnych atrybutów.

Zarządzanie systemem NIS jest zawsze scentralizowane i realizowane przez głównego administratora systemu. W przypadku usługi LDAP dane mogą być rozdystrybuowane na wiele serwerów i niezależnie zarządzane, nadal sprawiając wrażenie zintegrowanego systemu.



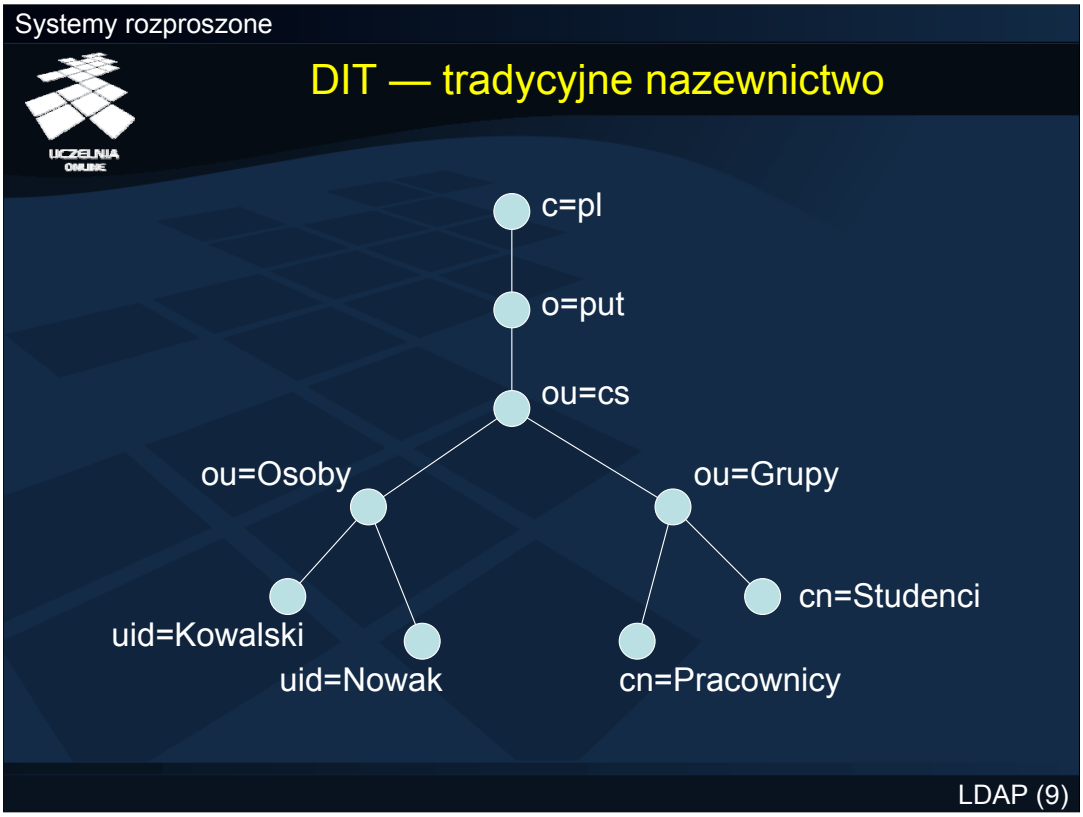
Podsumowując poprzednie rozważania można powiedzieć, że usługa katalogowa charakteryzuje się wieloma istotnymi zaletami. Umożliwia centralne zarządzanie lub scentralizowany ogląd konfiguracji całego systemu. Centralizacja jakkolwiek jest kojarzona negatywnie w przypadku systemów rozproszonych, to z punktu widzenia administratora jest własnością bardzo pożądaną.

Usługa katalogowa umożliwia dystrybucję odpowiedzialności za zarządzanie poszczególnymi fragmentami drzewa katalogowego. Dystrybucja ta nie musi sprowadzać się do podziału informacji pomiędzy wiele niezależnie zarządzanych serwerów, ale może być realizowana w ramach pojedynczego serwera.

Usługi katalogowe oferują dużą efektywność dostępu do danych, co jest koniecznością ponieważ odwołania do nich mogą być bardzo częste. W przypadku mocno obciążonych serwerów można zastosować zwielokrotnianie (replikację). Dostęp do danych np. konfiguracyjnych jest dużo bardziej efektywny niż w przypadku plików płaskich jak to ma miejsce w większości systemów uniksowych.

Interakcja klienta z serwerem podlega pewnym ograniczeniom co może być wykorzystane dla wymuszenia określonego uporządkowania informacji w bazie katalogowej. Dane wprowadzane do bazy muszą spełniać określone wymagania syntaktyczne, podlegające automatycznej kontroli.

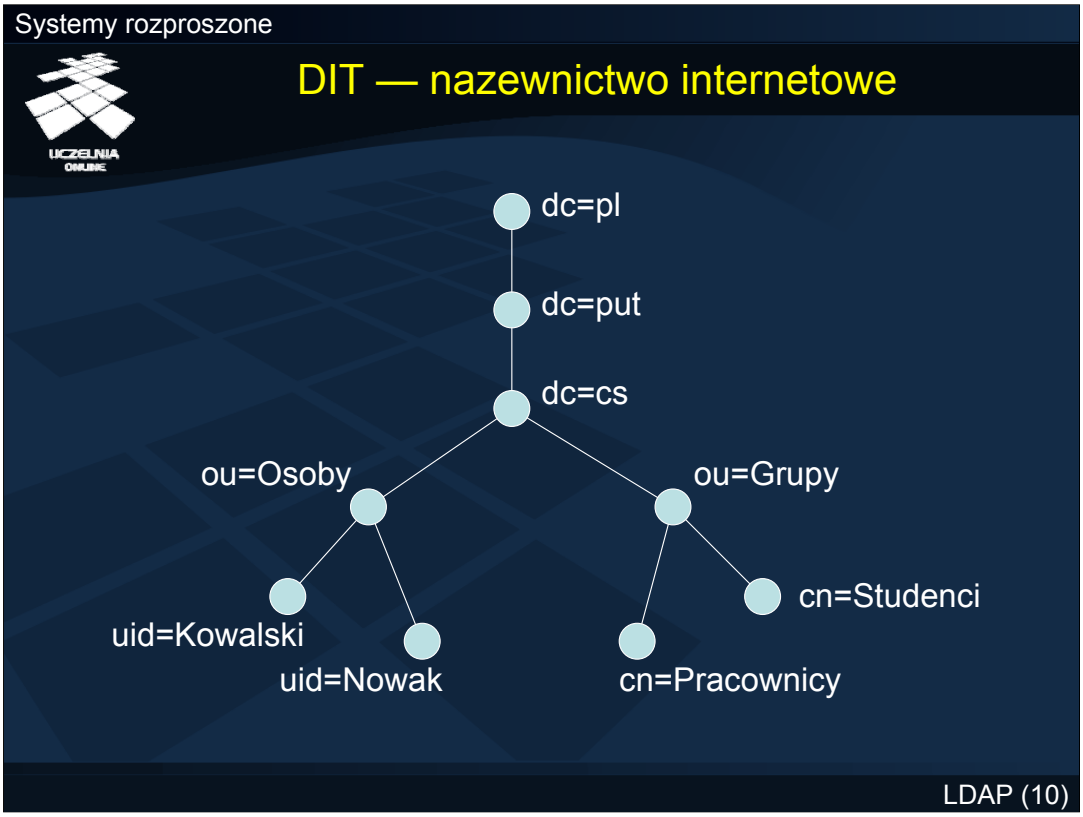
Dodatkowym atutem usług katalogowych jest możliwość zabezpieczenia komunikacji z serwerem. Powoduje to, że usługę tą można wykorzystywać nie tylko w izolowanych sieciach lokalnych, ale też i na większą skalę.



Drzewo informacji katalogowej może mieć bardzo różną strukturę. Decydujące w tym względzie są potrzeby lokalnej instytucji i rodzaj informacji jaka ma być przechowywana w bazie. Wewnętrzna struktura drzewa katalogowego musi być tak zaprojektowana, żeby była przejrzysta i wygodna w stosowaniu. Jeżeli przykładowa instytucja ma wyraźną strukturę organizacyjną, to prawdopodobnie kształt tej struktury powinien być odzwierciedlony w drzewie. Ułatwia to ewentualną dystrybucję odpowiedzialności na jednostki podrzędne poprzez przekazanie prawa do modyfikacji całego poddrzewa struktury katalogowej.

Na najwyższym poziomie hierarchia drzewa informacji katalogowej jest budowana najczęściej na dwa sposoby. Pierwszy, tradycyjny, jest zaprezentowany na slajdzie. Wykorzystano w nim węzły c (ang. *country* – państwo), o (ang. *organization* – organizacja) i ou (ang. *organizational unit* — jednostka organizacyjna). Drzewo takie zaczyna się więc od konkretnego węzła, który zawsze może być zintegrowany z innym drzewem poprzez wskazanie na serwer wyższego poziomu.

Hierarchia niższego poziomu, związana z budową drzewa bezpośrednio w danej organizacji korzysta z zagnieżdżonych węzłów ou, a na najniższym poziomie są węzły reprezentujące użytkowników (uid – ang. *user identifier*) i grupy (cn – ang. *common name*).



Alternatywną strukturą do tej przedstawionej na poprzednim slajdzie jest struktura bazująca na nazwach domenowych systemu DNS. W strukturze ten zastosowano węzły dc (ang. *domain component*), określające pojedynczy poziom w hierarchii nazw systemu DNS. Struktura tego typu mogłaby w przyszłości zastąpić system DNS, o ile przechowywałaby obiekty reprezentujące adresy IP i inne dane z systemu DNS. Hierarchia niższego poziomu jest w tym przypadku identyczna jak w podejściu tradycyjnym.

Drzewo informacji katalogowej

Podobieństwo do systemu plików, ale:

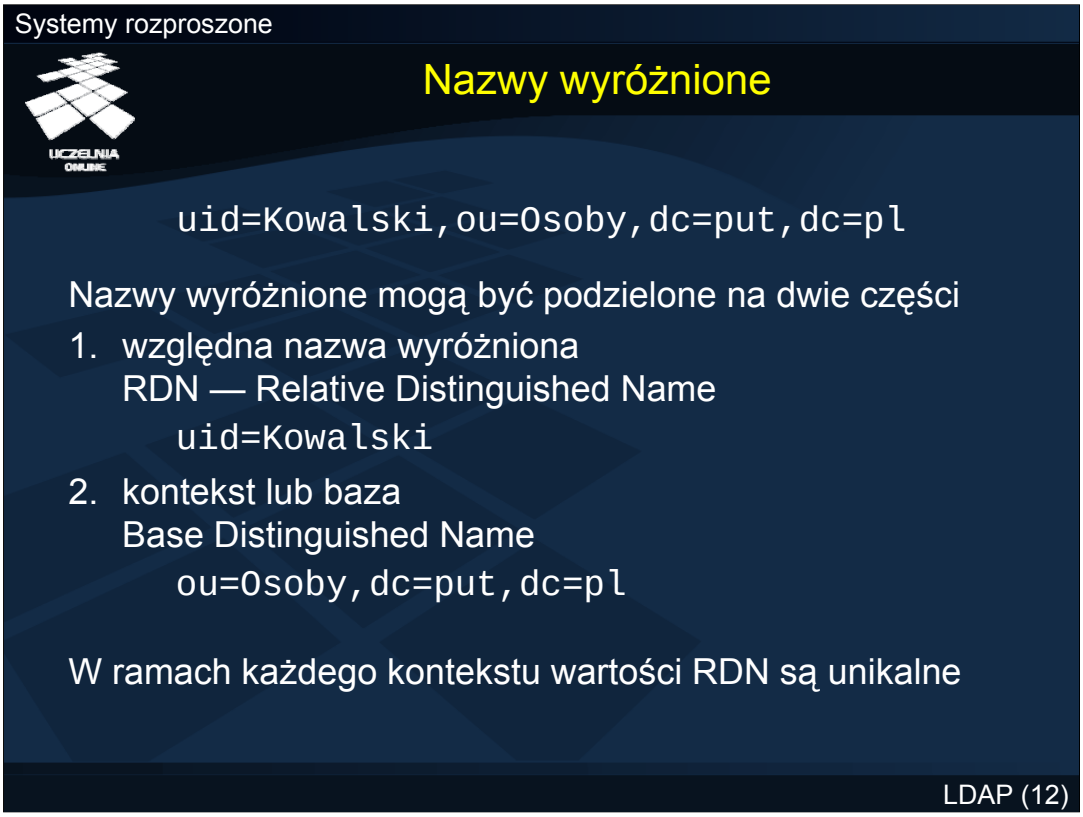
- brak korzenia
- każdy węzeł drzewa DIT może jednocześnie zawierać dane i posiadać węzły potomne
- możliwa odwrotna struktura nazw obiektów w drzewie:

```
/usr/local/bin/pico
/dc=pl/dc=put/ou=Osoby/uid=Kowalski
uid=Kowalski,ou=Osoby,dc=put,dc=pl
```


Drzewo informacji katalogowej może być kojarzone z systemem plików typowego systemu operacyjnego. Istnieją jednak pewne dość istotne różnice pomiędzy tymi hierarchiami. Po pierwsze w drzewie informacji katalogowej nigdy nie ma korzenia. Węzeł najwyższego poziomu, np. c=pl, jest po prostu jednym z węzłów, ale jego pozycja w katalogu może zawsze zostać „zdegradowana” poprzez nadbudowę węzłem wyższego poziomu. Odwołania do usługi katalogowej zawsze zawierają wskazanie na węzeł bazowy, od którego rozpoczyna się wykonywanie bieżących operacji. W przypadku systemów plików istnieje zawsze katalog główny, który pozostaje główny podczas całej pracy systemu. Stworzenie katalogu wyższego poziomu wymuszałoby konieczność rekonfiguracji aplikacji klienckich. Tego typu problemy występują np. w rozproszonych systemach plików podczas integracji kilku istniejących hierarchii plików.

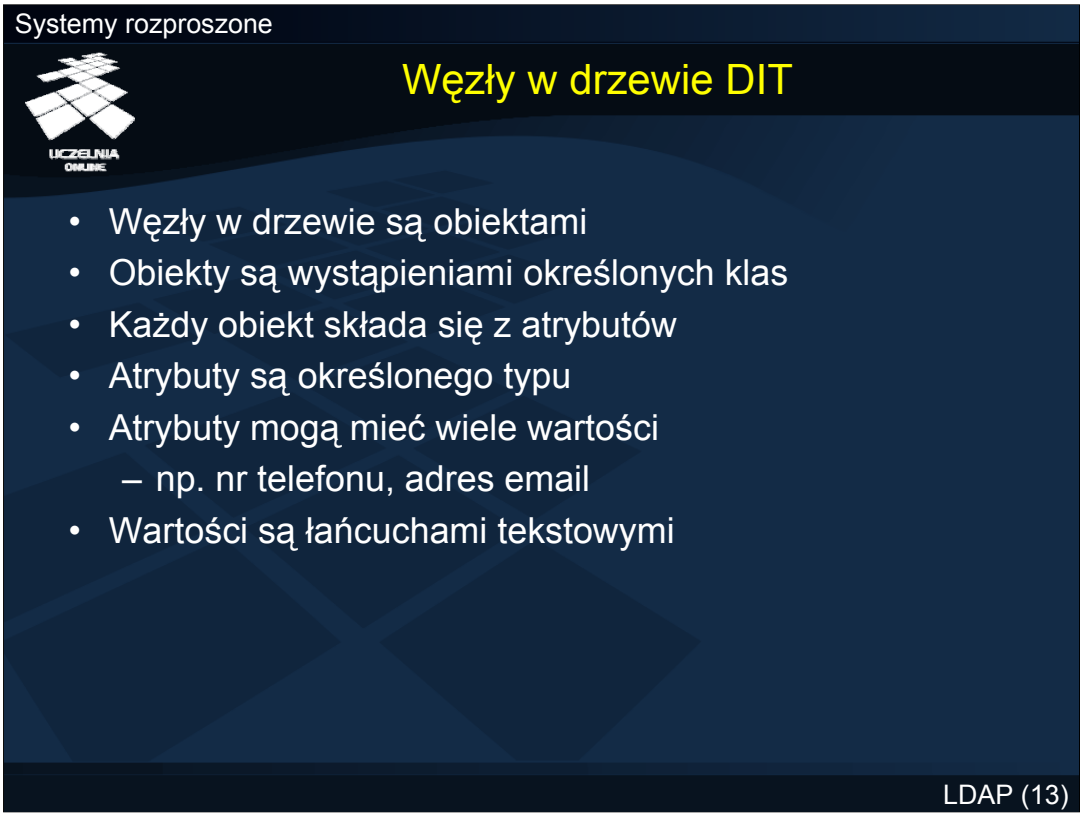
Druga istotna różnica polega na rozróżnieniu katalogów i plików w systemie plików. Katalogi służą do reprezentowania struktury katalogowej a pliki do przechowywania danych. W przypadku usługi X.500 każdy węzeł może pełnić rolę katalogu i pliku jednocześnie. Obiekty mają bowiem przypisywane atrybuty, które opisują je (są danymi) ale te same obiekty mogą też posiadać obiekty niższe w hierarchii.

Zapis pełnych nazw obiektów może też stosować odwrotną kolejność reprezentacji hierarchii. W przedstawionym przykładzie plik *pico* znajduje się w katalogu */usr/local/bin*; katalog najwyższego poziomu (*/usr*) znajduje się w tym przypadku skrajnie po lewej. Podobna sytuacja występuje w tekstowej reprezentacji nazw Microsoft Active Directory (przykład 2). Ale już w przypadku wielu serwerów LDAP stosuje się zapis z ostatniego przykładu, gdzie po lewej stronie jest nazwa węzła najniższego poziomu.



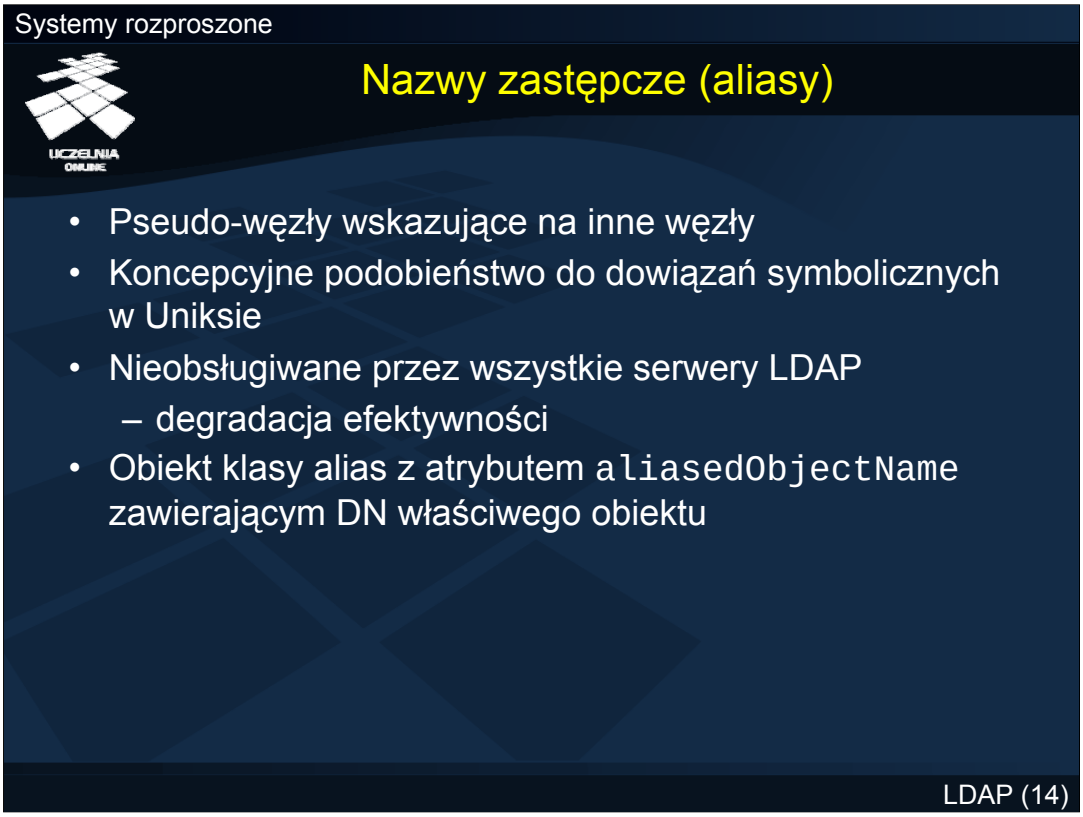
Identyfikacja obiektów w drzewie informacji katalogowej jest realizowana poprzez **nazwy wyróżnione** (ang. *distinguished names*). Nazwa wyróżniona powstaje poprzez złożenie **względnych nazw wyróżnionych** (ang. *relative distinguished name – RDN*) od bieżącego węzła aż do węzła najwyższego poziomu w drzewie. Dla każdego węzła można więc wskazać jego RDN jako pierwszy (lub ostatni, zależnie od przyjętej konwencji zapisu) człon nazwy i tzw. **kontekst** (lub bazę), która jest nazwą wyróżnioną węzła nadrzędnego (katalogu). Wartości RDN w ramach każdego kontekstu muszą być unikalne, co w konsekwencji gwarantuje globalną unikalność wszystkich nazw wyróżnionych w drzewie.

Pojęcie nazwy wyróżnionej jest konsekwencją tego, że obiekt może mieć kilka nazw zapisanych w różnych atrybutach, ale tylko jedna z tych nazw pełni rolę identyfikującą i musi spełniać warunek unikalności w ramach kontekstu. Możemy mieć więc w systemie kilku Janów Kowalskich, ponieważ będą się posługiwali innymi identyfikatorami, np. uid=jan i uid=janek.

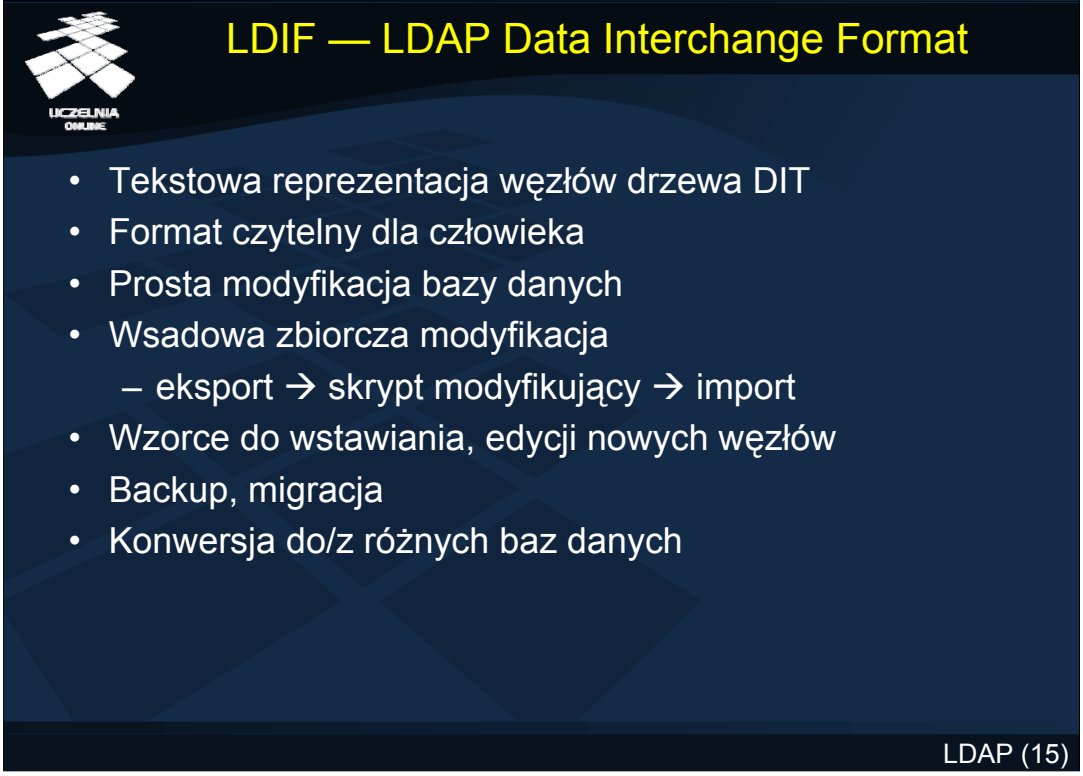


Drzewo informacji katalogowej ma postać hierarchicznego zbioru obiektów-węzłów. Każdy obiekt jest wystąpieniem określonej klasy lub kilku klas. Obiekty składają się ze zbioru nazwanych atrybutów. Każdy atrybut ma swój ściśle określony typ. Cechą specyficzną drzewa DIT jest to, że atrybuty w obiektach mogą być wielowartościowe. Jest to kolejna istotna różnicą w stosunku do relacyjnych baz danych, gdzie przeprowadza się normalizację danych polegającą między innymi na usunięciu atrybutów wielowartościowych. Z praktycznego punktu widzenia jednak atrybuty wielowartościowe są jak najbardziej użyteczne. Klasycznym przykładem może tu być numer telefonu. Każdy użytkownik może być dostępny pod kilkoma numerami (służbowy, domowy, komórkowy) i wszystkie te numery można umieścić w definicji jednego obiektu.

Wartości atrybutów reprezentowane są w postaci łańcuchów tekstowych. Nie oznacza to jednak całkowitej swobody przypisywania im dowolnej wartości, bo atrybuty mają swoje typy.



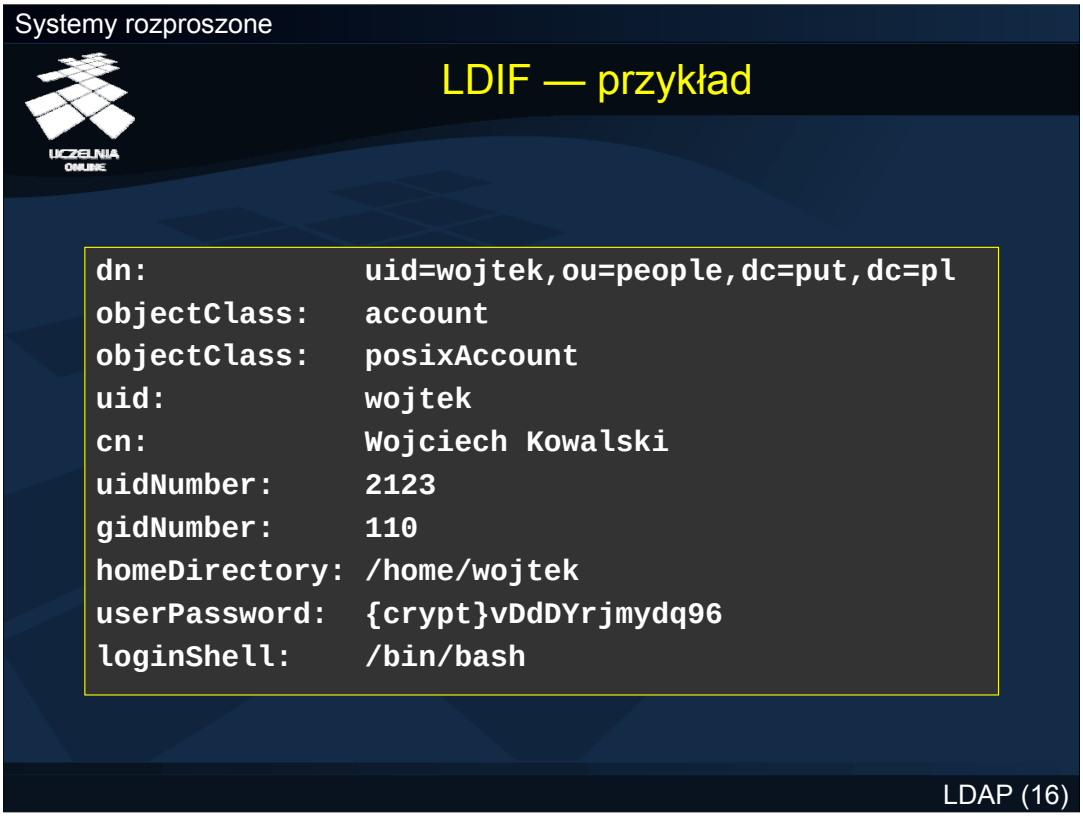
W drzewie informacji katalogowej mogą być przechowywane specjalne węzły nie przechowujące nowej informacji a jedynie wskazujące na inne istniejące w drzewie węzły. Są to węzły definiujące **nazwy zastępcze** (aliasy) dla istniejących węzłów. Jest to koncepcyjna analogia do dowiązań symbolicznych w systemie plików Uniksa. Węzły takie posiadają swoją nazwę wyróżnioną po to by móc być jednoznacznie identyfikowanymi w drzewie, ale jako dane przechowują nazwę wyróżnioną docelowego węzła, na który wskazują. Odwołanie do węzła przechowującego nazwę zastępczą powoduje odczytanie docelowej nazwy wyróżnionej i dalsze przeszukiwanie w celu odnalezienia docelowego obiektu. Działanie takie oczywiście powoduje pewien spadek efektywności odwołań do obiektów ponieważ lokalizacja obiektów jest realizowana wielokrokowo. Z drugiej jednak strony pozwala to na umieszczenie odniesień do obiektów w różnych fragmentach drzewa reprezentujących różne aspekty występowania obiektu.



LDIF to standardowy format reprezentacji danych przechowywanych w drzewie informacji katalogowej. Jest to format tekstowy co jest jego dużą zaletą, ponieważ jest dzięki temu czytelny dla użytkownika i może być przetwarzany przy użyciu zwykłych edytorów tekstowych. Posługiwanie się tym formatem umożliwia modyfikację zawartości drzewa. Modyfikacje realizowane w ten sposób są bardzo przydatne w przypadku wprowadzania istotnych aktualizacji dotyczących wielu węzłów. Przeprowadzenie aktualizacji polega na wyeksportowaniu wszystkich danych, ich przetworzeniu wsadowym operując bezpośrednio na tekście, i ostatecznie na ponownym wprowadzeniu do serwera.

Format LDIF jest przydatny również podczas wykonywania wsadowych, programowych operacji przetwarzania zawartości drzewa DIT. W formacie tym można bowiem zapisać szablony dla nowotworzonych obiektów lub szablony umożliwiające przeprowadzenie standardowych modyfikacji obiektów.

Kolejnym zastosowaniem standardu LDIF jest tworzenie kopii zapasowych DIT. Kopia taka jest bezpieczna, ponieważ jest reprezentowana w postaci tekstowej. Kopię taką można też wykorzystać do przeniesienia całej bazy do serwera innego producenta, ponieważ LDIF powinien być rozumiany przez każdy serwer LDAP.

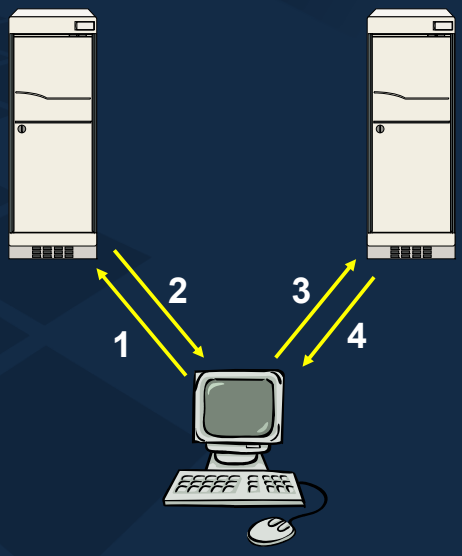


Slajd przedstawia przykładowy fragment drzewa DIT w formacie LDIF. Fragment ten definiuje pojedynczy obiekt reprezentujący użytkownika. Każda linia pliku definiuje pojedynczy atrybut, którego nazwa znajduje się na początku, a po dwukropku wartość. Pierwszym atrybutem jest dn czyli nazwa wyróżniona obiektu. Poniżej znajdują się atrybuty określające klasę obiektu. W tym przypadku obiekt jest wystąpieniem dwóch klas: *accout* i *posixAccount*. Pozostałe atrybuty zawierają informację o użytkowniku: jego identyfikator systemowy, nazwisko, identyfikator numeryczny, numer grupy, nazwę katalogu domowego, hasło i nazwę domyślnego interpretera poleceń.

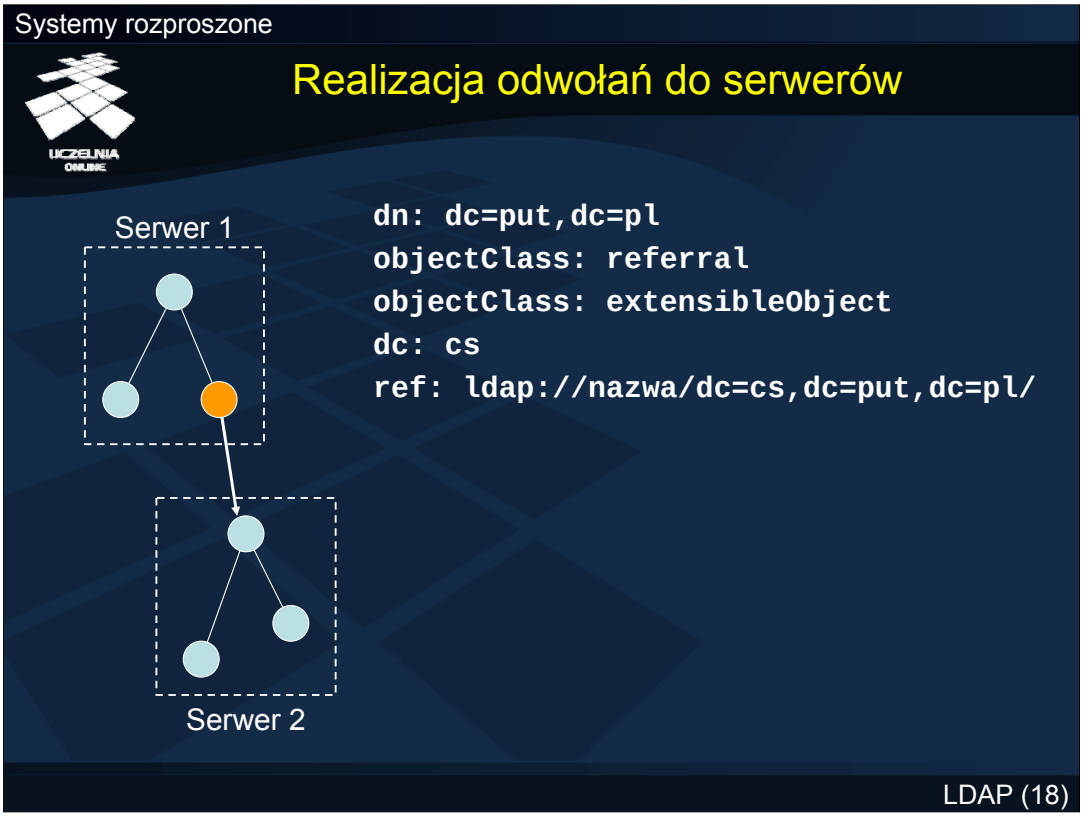
Definicje kolejnych obiektów w formacie LDIF oddzielane są od siebie pustą linią.

Odwołania do innych serwerów

1. zapytanie klienta
2. zwrot adresu innego serwera
3. drugie zapytanie klienta
4. odpowiedź

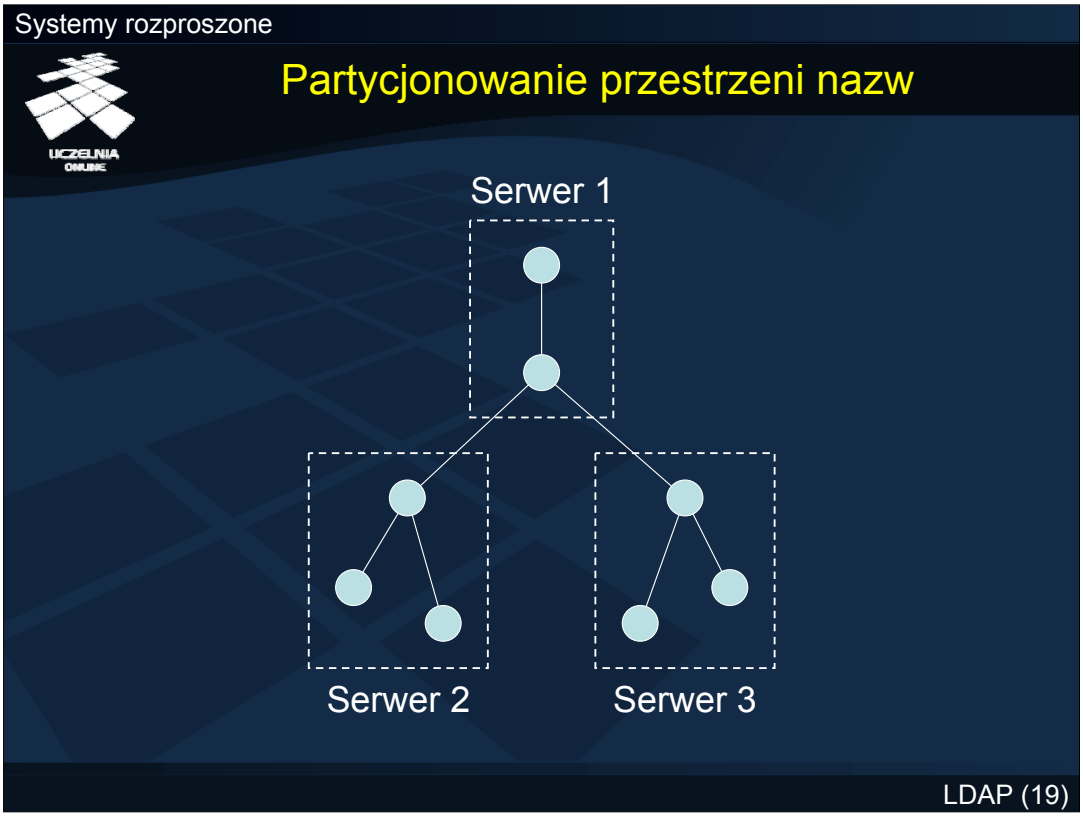


Integrację danych zawartych w drzewach informacji katalogowej różnych serwerów można przeprowadzić poprzez zastosowanie mechanizmu odwołań. Klient wysyła swoje zapytanie do serwera domyślnego czy najbliższego (krok 1). Serwer ten stwierdza jednak, że nie przechowuje lokalnie informacji poszukiwanych przez klienta, ale wie który serwer ją przechowuje i odsyła klientowi wskazanie na ten serwer (2). Klient wykonuje połączenie do drugiego serwera (3) i uzyskuje poszukiwaną informację (4). Działanie to przypomina iteracyjny tryb odpytywania serwerów DNS, które w podobny sposób naprowadzały na siebie.



Odwołania do innych serwerów są zrealizowane podobnie jak aliasy. W drzewie informacji katalogowej, które ma zawierać odwołanie do innego serwera, umieszcza się specjalny obiekt klasy *referral*, który posiada atrybut *ref* przechowujący adres URL docelowego serwera LDAP. Jest to specjalny węzeł, do którego dotarcie powoduje przesłanie informacji zwrotnej do klienta o docelowym miejscu poszukiwania informacji.

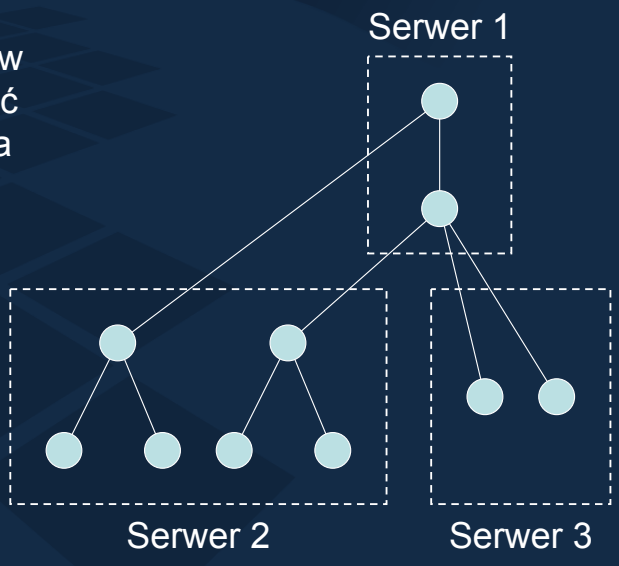
Węzeł będący odwołaniem do innego serwera ma swoją nazwę wyróżnioną tak jak każdy inny węzeł. Nazwa ta jest identyczna z nazwą węzła, na który wskazuje.



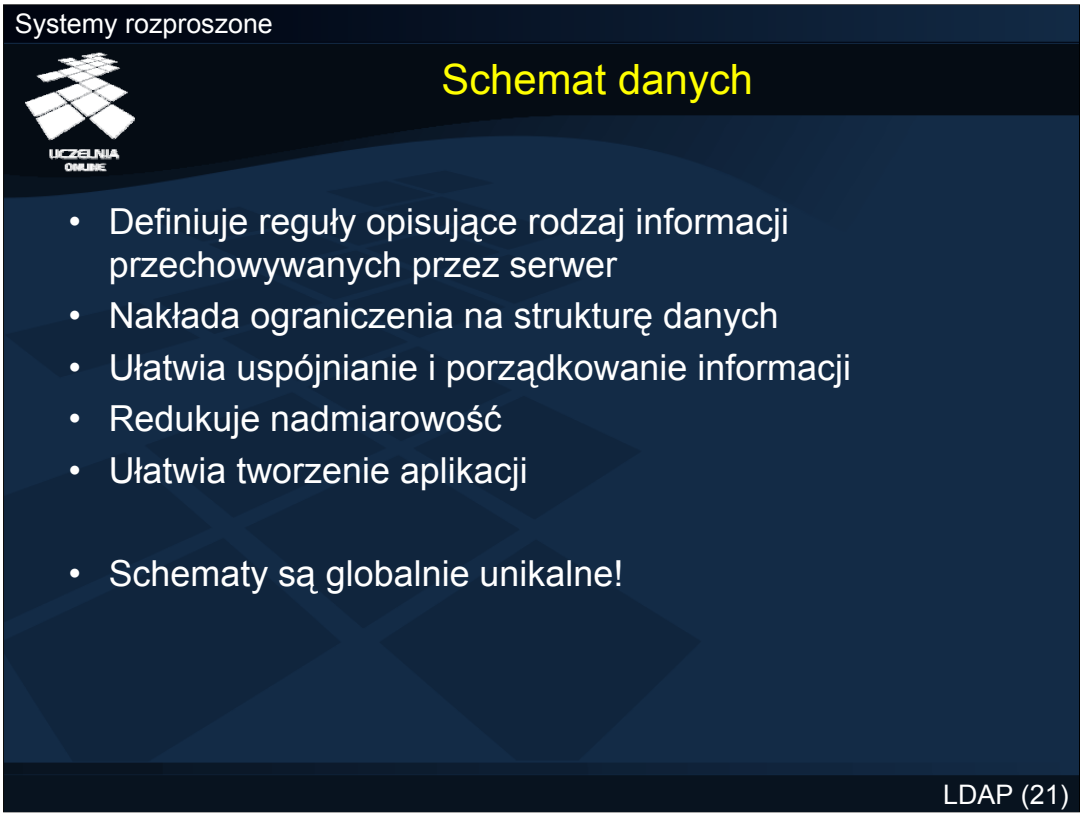
Przestrzeń nazw reprezentowanych w usłudze katalogowej można rozdzielić (partycjonować) na kilka serwerów. Działanie takie ma na celu usytuowanie danych bliżej użytkowników, którzy głównie z nich korzystają. Z drugiej strony może to też spowodować zwiększenie niezawodności, gdyż awaria serwera jednej partycji nie zablokuje dostępu do innej. Rysunek pokazuje przykład podziału drzewa na 3 serwery.

Błędne partycjonowanie

- Wszystkie obiekty w partycji muszą mieć wspólnego przodka
- Wspólny przodek musi należeć do partycji

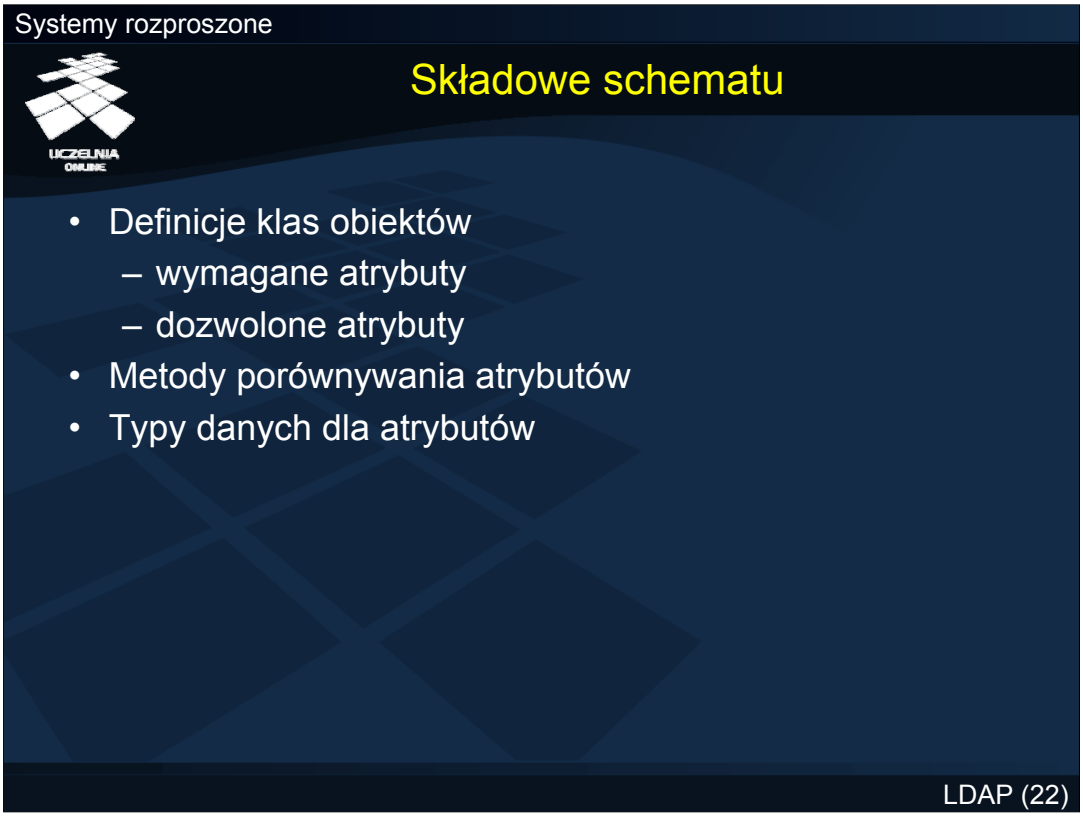


Nie każde partycjonowanie jest poprawne. W partycji na serwerze 2 nie wszystkie węzły mają wspólnego przodka. Mam w tym przypadku do czynienia *de facto* z dwoma partycjami. Serwer 3 posiada węzły, które co prawda mają wspólnego przodka, ale ten przodek nie znajduje się w partycji.

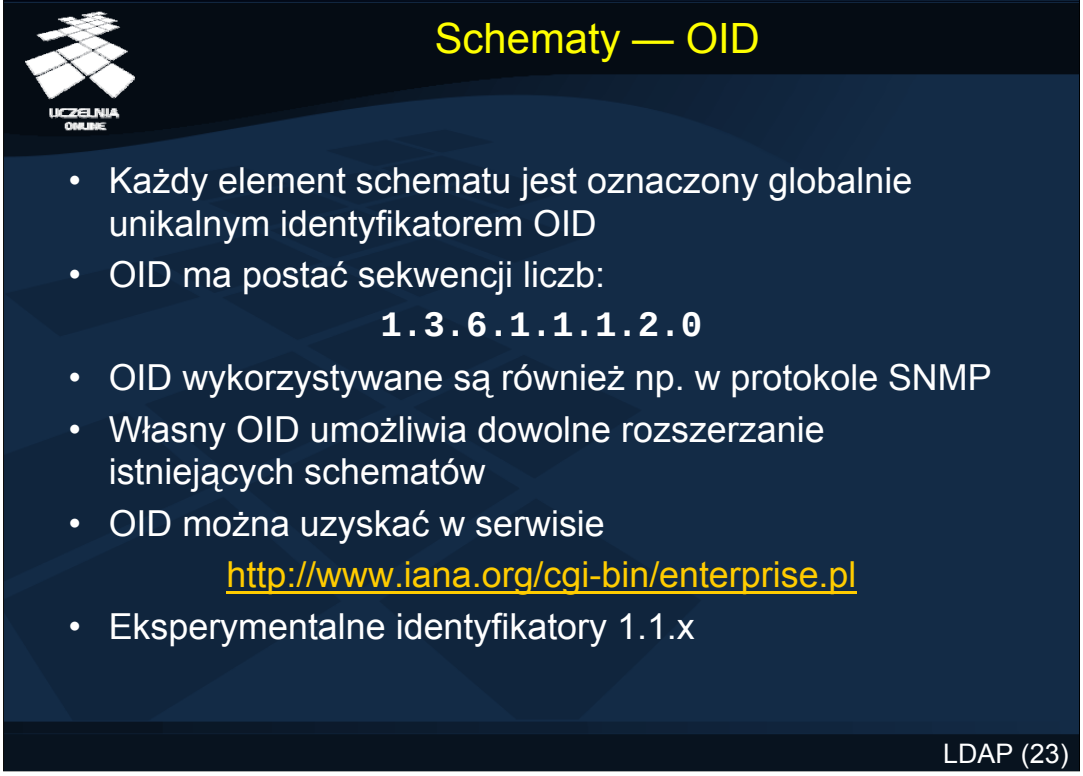


Dane przechowywane na serwerach LDAP podlegają pewnym ograniczeniom opisanym przez **schemat danych**. Stosownie schematu porządkuje współpracę różnych aplikacji z usługą katalogową. Jednocześnie redukowana jest możliwa nadmiarowość wynikająca z różnych metod reprezentacji tej samej informacji. Twórcy aplikacji mają również ułatwione zadanie gdy schematy danych są dostępne, ponieważ mogą poczynić pewne założenia upraszczające co do organizacji danych.

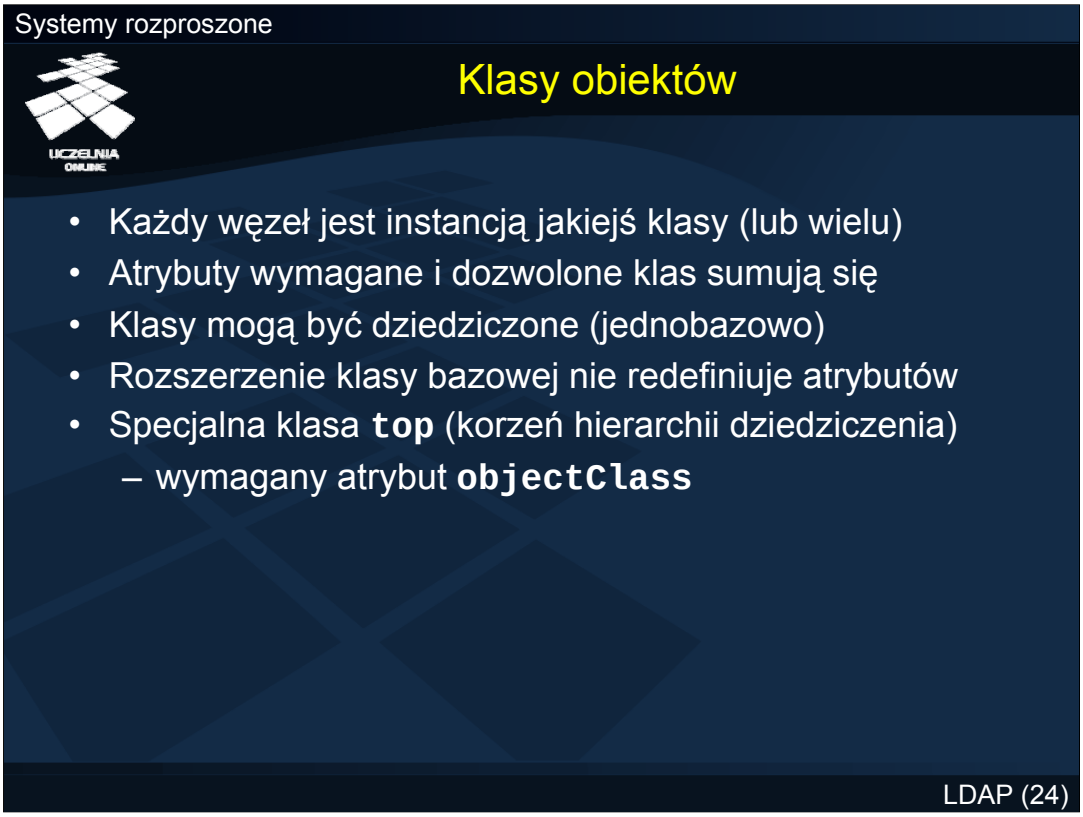
Schematy danych są globalnie unikalne. Oznacza to, że każdy schemat ma swój identyfikator, który jest unikalny i jednoznacznie identyfikowany przez wszystkie serwery na świecie. Podejście to ma na celu umożliwienie przeprowadzenia w przyszłości integracji niezależnych drzew informacji katalogowych.



Schemat danych zawiera definicje klas obiektów, które będą przechowywane na serwerze. Nie można wprowadzić do serwera obiektu, który ma niezdefiniowaną klasę (czyli typ). Ponieważ obiekty składają się z atrybutów, definicja klasy zawiera listę atrybutów, które są wymagane i które są dozwolone. Każdy atrybut ma swój typ, który podobnie jak klasy jest jednoznacznie identyfikowany. Dla każdego typu atrybutu definiuje się również operatory przetwarzające je.

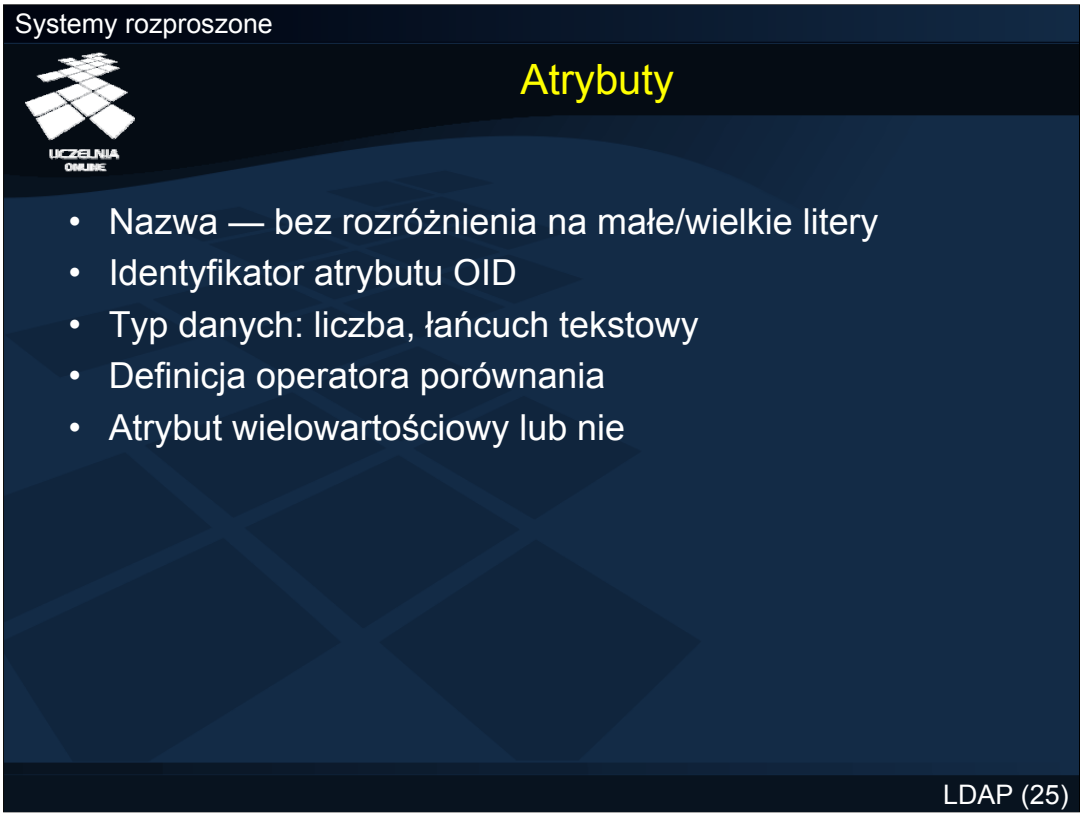


Globalną unikalność schematów uzyskuje się poprzez oznaczenie wszystkich klas, atrybutów i operatorów unikalnymi identyfikatorami **OID** (ang. *object identifier*). Identyfikatory te tworzone są poprzez sklejanie ze sobą kolejnych wartości liczbowych. Liczby te reprezentują pewnego rodzaju hierarchię analogicznie jak podsieci w systemie adresacji IP. Każda instytucja może wystąpić o nowy, globalnie unikalny identyfikator dla swoich potrzeb. Identyfikator ten można następnie traktować jako prefiks dla tworzenia nowych, unikalnych identyfikatorów szczegółowych. Podejście to gwarantuje globalną unikalność wszystkich identyfikatorów na świecie. Do eksperymentów można wykorzystać identyfikatory zaczynające się od 1.1.



Każdy węzeł znajdujący się w drzewie informacji katalogowej jest wystąpieniem jakiejś klasy lub wielu klas zdefiniowanych w schemacie. Jeżeli obiekt jest wystąpieniem wielu klas, to odpowiednie atrybuty wymagane i opcjonalne poszczególnych klas się sumują.

Klasy mogą być dziedziczone. Klasa pochodna może rozszerzyć definicję klasy bazowej, ale nie może jej zmienić. Możliwe jest tylko dziedziczenie jednobazowe. Na szczycie hierarchii dziedziczenia znajduje się specjalna klasa *top*, której jedynym atrybutem (wymaganym) jest atrybut *objectClass*, wskazujący na nazwę klasy, której dany obiekt jest wystąpieniem. Atrybut ten jest więc wymagany w definicjach wszystkich klas.

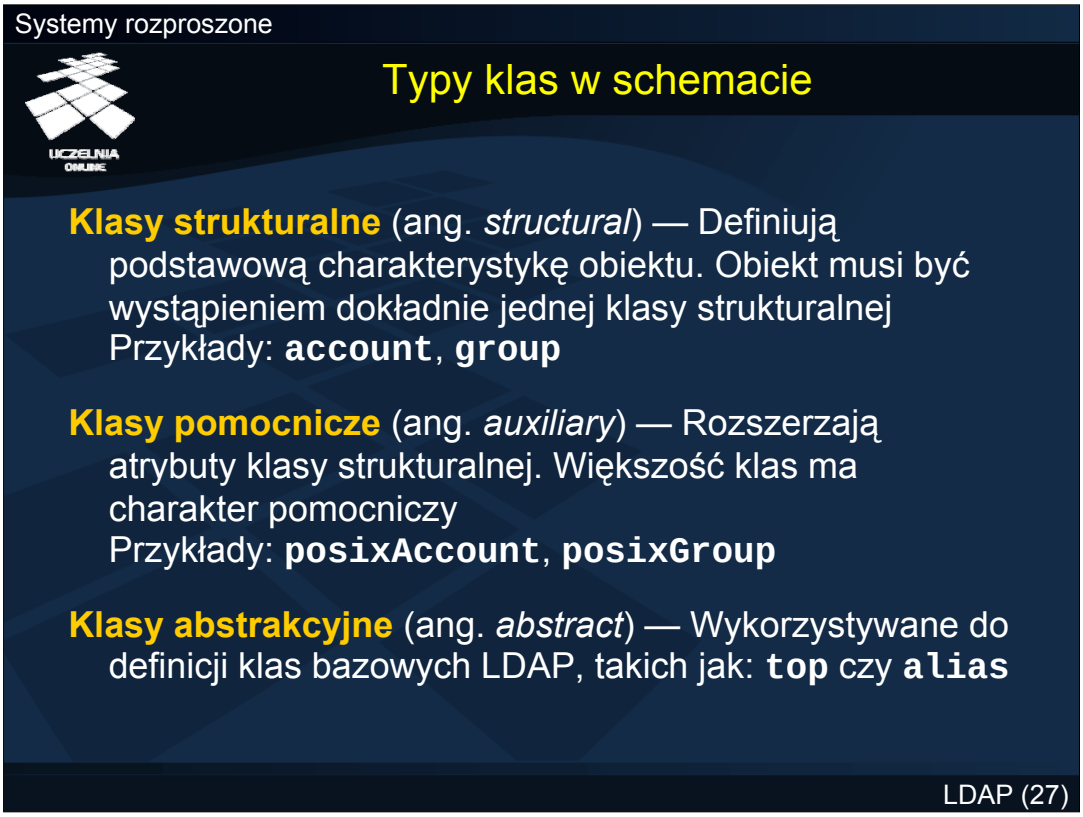


W nazwach atrybutów nie rozróżnia się małych i wielkich liter. Atrybuty, podobnie jak klasy są jednoznacznie identyfikowane poprzez unikalny identyfikator OID. Każdy atrybut ma przypisany typ danych, jak również operatory porównania. W definicji atrybutu znajduje się również informacja czy atrybut może przechowywać wiele wartości. W zapisie LDIF atrybut wielowartościowy jest wymieniony kilkukrotnie z różnymi wartościami.

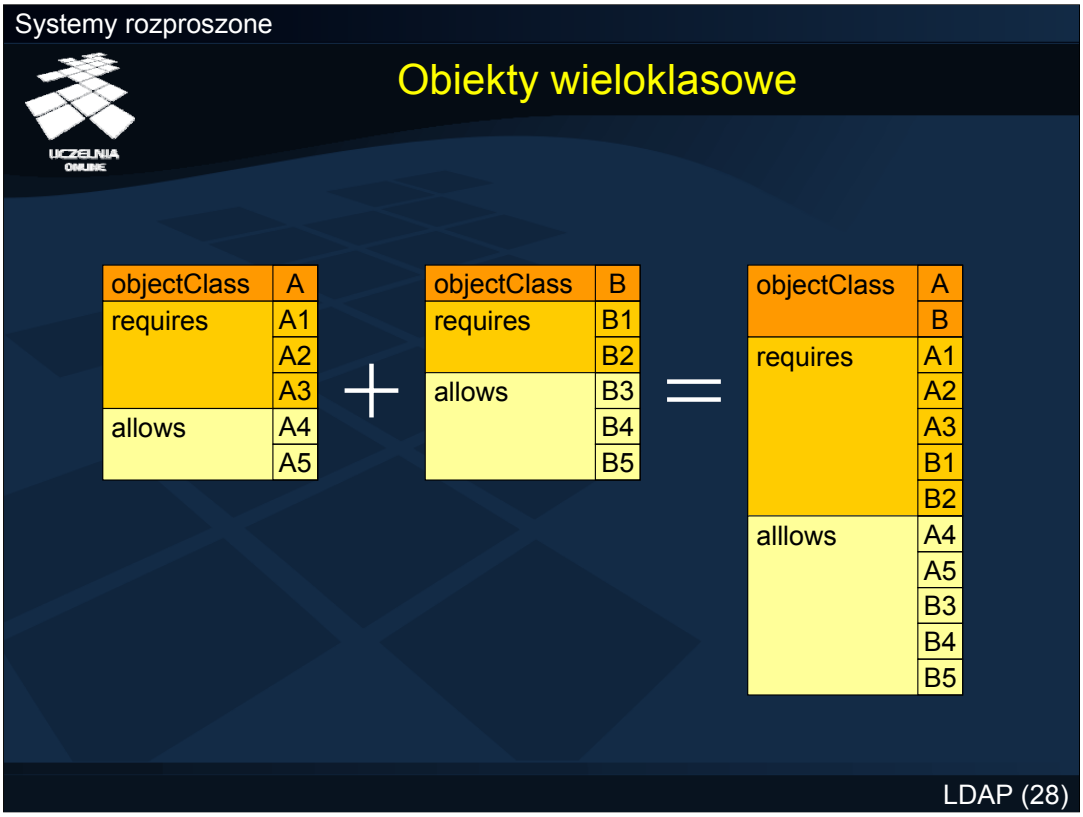
Przykłady atrybutów

Nazwa skrócona	Nazwa pełna	Opis
`uid`	User identifier	Identyfikator użytkownika
`cn`	Common name	Nazwa
`sn`	Surname	Nazwisko
`ou`	Organizational Unit	Jednostka organizacyjna
`o`	Organization	Jednostka
`dc`	Domain Component	Składnik nazwy domenowej
`c`	Country	Państwo

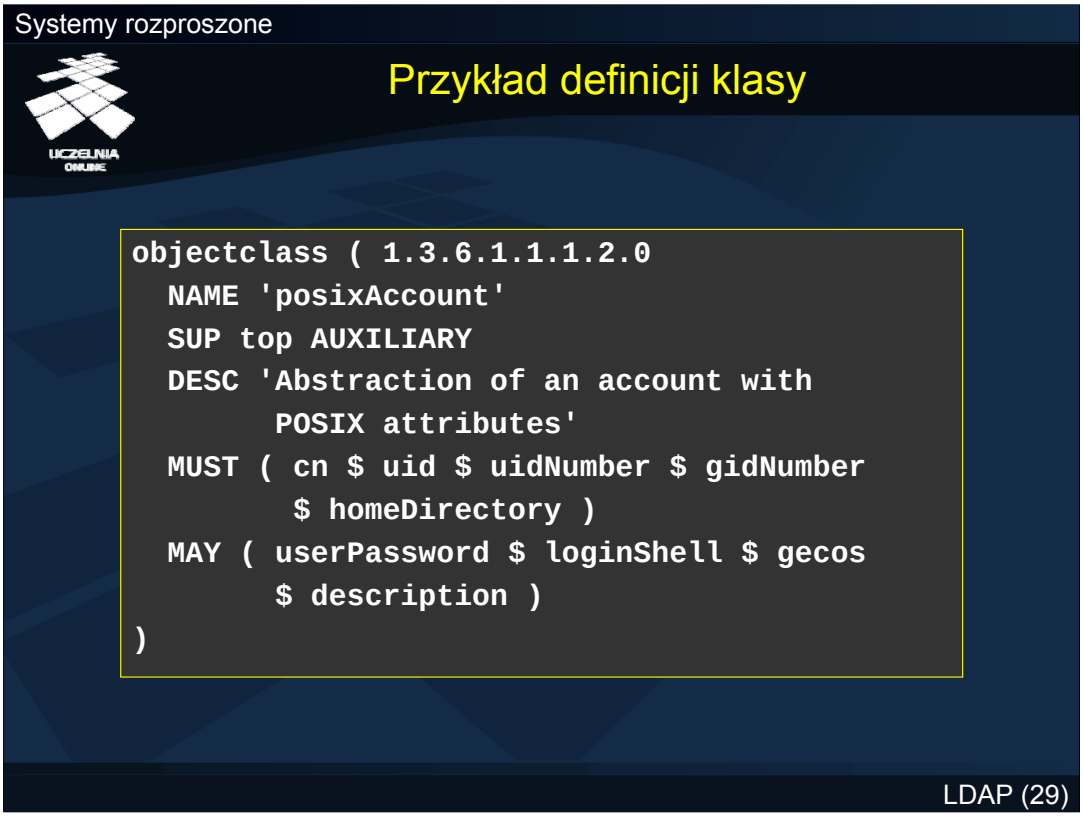
Tabelka przedstawia kilka standardowych, często stosowanych atrybutów. Atrybuty te zostały zdefiniowane w schematach danych, które są najczęściej wykorzystywane w Internecie i które zostały ustandaryzowane poprzez odpowiednie dokumenty RFC. Tworząc swoją bazę informacji katalogowej warto w miarę możliwości stosować standardowe klasy i atrybuty, ewentualnie rozszerzając je o specyficzne pozycje. Podejście takie ułatwi integrację usługi katalogowej z aplikacjami i innymi serwerami.



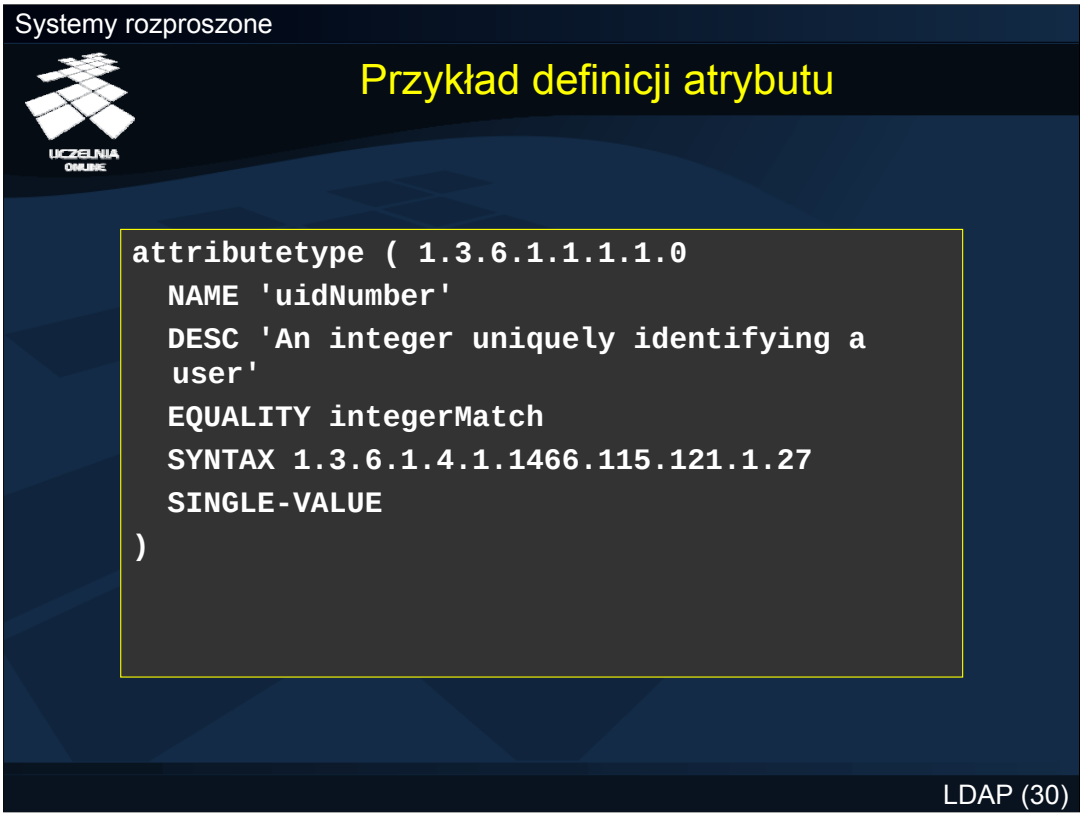
Wszystkie klasy wykorzystywane w usłudze LDAP dzielą się na: klasy strukturalne, pomocnicze i abstrakcyjne. Każdy obiekt musi być wystąpieniem dokładnie jednej klasy strukturalnej. **Klasa strukturalna** określa *charakter* obiektu, np. konto w systemie czy komputer. Atrybuty przewidziane w klasie strukturalnej mogą być jednak niewystarczające do pełnego opisu własności obiektu, stąd stosuje się dodatkowo **klasy pomocnicze**, które umożliwiają zdefiniowanie dodatkowych atrybutów przypisywanych obiektowi. Każdy obiekt może być wystąpieniem dowolnej liczby klas pomocniczych. Istnieją też **klasy abstrakcyjne**, które są klasami specjalnymi, umożliwiającymi definicję takich klas jak *top* czy *alias*.



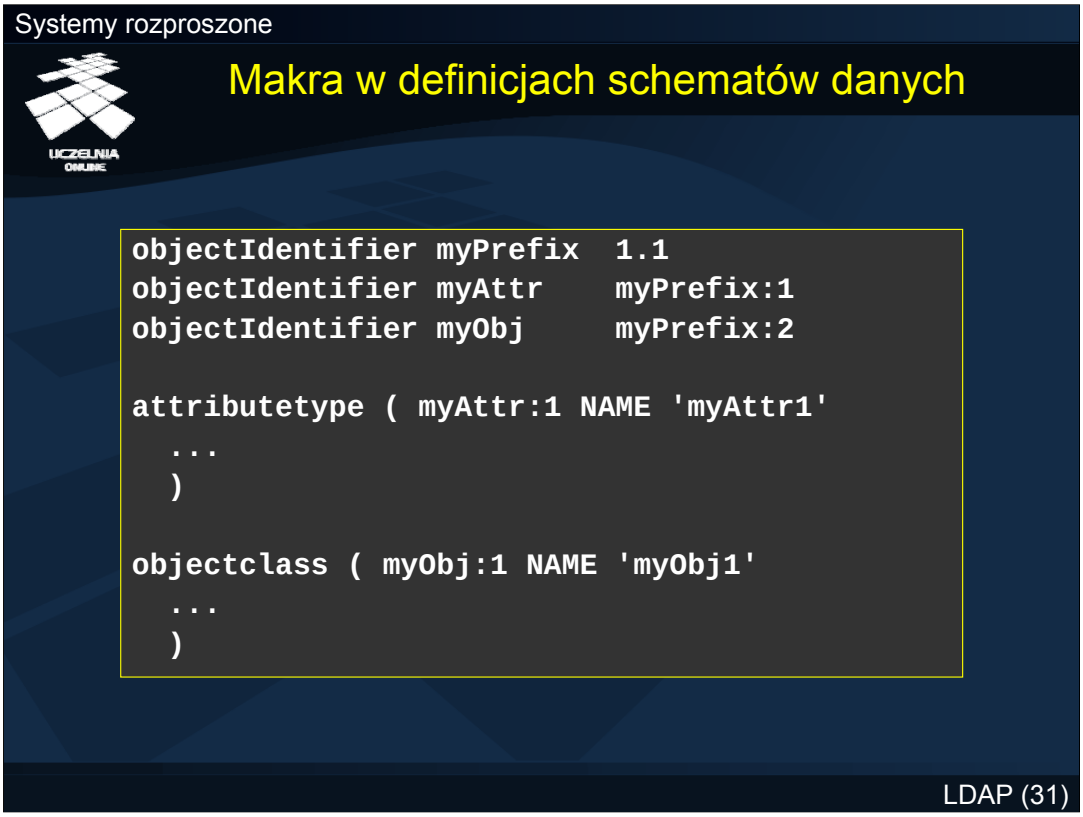
Rysunek pokazuje w jaki sposób powstaje obiekt będący instancją wielu klas. Załóżmy, że klasa A jest klasą strukturalną i wymaga zdefiniowania atrybutów A1, A2 i A3 oraz dopuszcza definicje atrybutów A4 i A5. Mamy też pomocnicza klasę B, która wymaga zdefiniowania atrybutów B1 i B2 oraz dopuszcza atrybuty B3, B4 i B5. Jeżeli obiekt będzie wystąpieniem klasy A i B, to w definicji takiego obiektu muszą być zawarte co najmniej wszystkie atrybuty wymagane przez klasy A i B, a więc w tym przypadku atrybuty A1, A2, A3, B1 i B2. Klasa może również definiować pozostałe atrybuty opcjonalne z klas A i B, czyli atrybuty A4, A5, B3, B4 i B5.



Slajd przedstawia przykładową definicję klasy zawartą w pliku opisujących konkretny schemat danych. Jest to definicja klasy *posixAccount*, która jest klasą pomocniczą, dziedziczącą z klasy *top*. W definicji klasy znajduje się jej opis (pole DESC). Następnie mamy wymienione atrybuty wymagane (MUST) i opcjonalne (MAY) klasy. Znak „$” służy do separacji poszczególnych atrybutów. Definicja klasy rozpoczyna się od identyfikatora OID.



Na podobnej zasadzie jak definicje klas w schemacie znajdują się definicje atrybutów. Na slajdzie przedstawiono definicję atrybutu *uidNumber*. Jest to atrybut liczbowy, na co wskazuje operator badający równość tego atrybutu; pole EQUALITY ma wartość *integerMatch*, oznaczające porównywanie liczb całkowitych. Dokładny typ atrybutu jest wskazany w polu SYNTAX. Atrybut *uidNumber* jest atrybutem jednowartościowym (SINGLE-VALUE).



Zarządzanie schematami danych, szczególnie tymi sporych rozmiarów, można zoptymalizować stosując makra do rozwijania początkowych łańcuchów składających się na identyfikatory OID. Slajd pokazuje definicję makra *myPrefix*, a następnie makr *myAttr* i *myObj* rozszerzających prefiks 1.1 o kolejne pola. Dzięki temu definicja atrybutu *myAttr1* będzie zawierać identyfikator 1.1.1.1, a definicja obiektu *myObj1* korzysta z identyfikatora 1.1.2.1. Prefiks 1.1 jest prefiksem eksperymentalnym. Po uzyskaniu prefiksu dla firmy w powyższym schemacie wystarczy jedynie zmienić definicję makra *myPrefix* i schemat będzie gotowy do użycia.

Typ danych	OID	Opis
Boolean	1.3.6.1.4.1.1466.115.121.1.7	Wartość logiczna
Distinguished name	1.3.6.1.4.1.1466.115.121.1.12	Nazwa wyróżniona
Directory string	1.3.6.1.4.1.1466.115.121.1.15	Łańcuch tekstowy UTF8
Integer	1.3.6.1.4.1.1466.115.121.1.27	Liczba całkowita
OID	1.3.6.1.4.1.1466.115.121.1.38	Identyfikator obiektu

Istnieje kilka standardowych schematów danych, definiujących m.in. podstawowe typy danych. Tabelka prezentuje kilka podstawowych typów danych takich, jak: wartości logiczne, nazwy wyróżnione, łańcuch tekstowe czy liczby. Niskopoziomowe identyfikatory tych typów są jak widać dość rozbudowane. Ich istnienie jednak pozwala na jednoznaczną weryfikacja tego, że każdy serwer tak samo interpretuje zapis *integer*.

Operatory

- porównanie (ang. *equality*)
- porządkowanie (ang. *ordering*)
- przeszukiwanie (ang. *substring*)

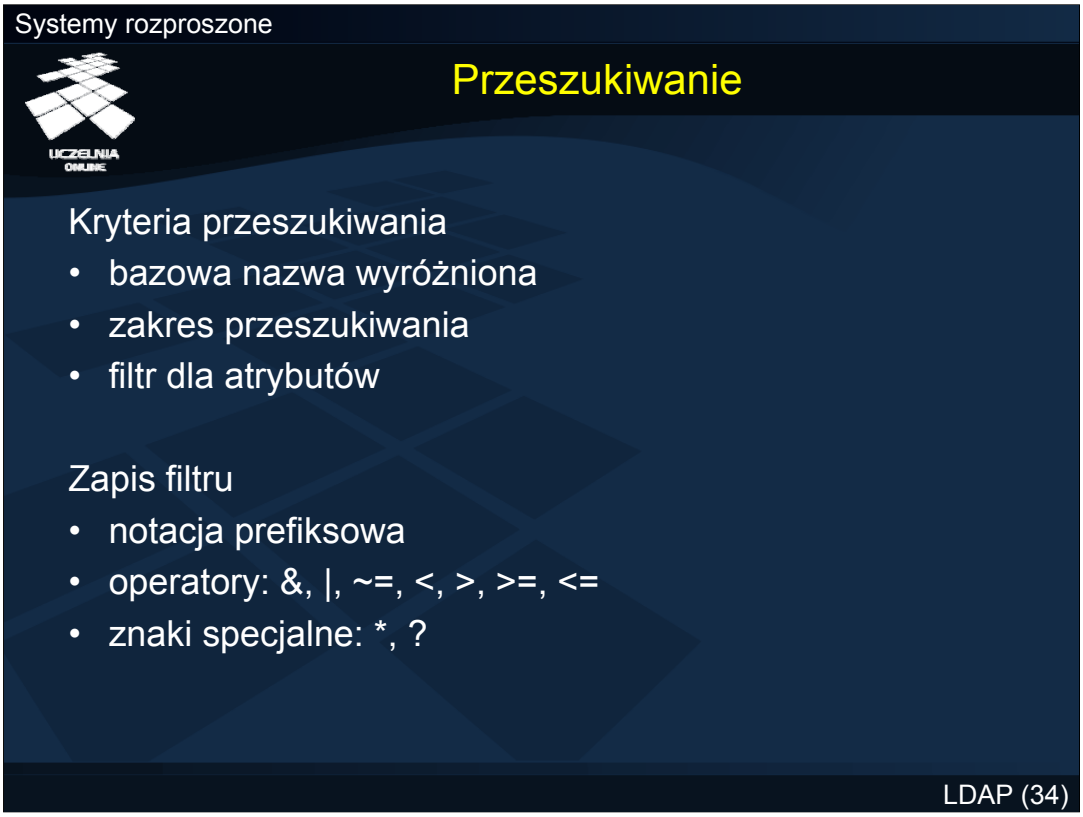
Nazwa operatora	Kontekst	Opis
booleanMatch	equality	Logiczna równość
objectIdentiferMatch	equality	Równość identyfikatorów
numericStringMatch	equality	Równość wartości liczbowych
numericStringOrdering	ordering	Kolejność sortowania
numericStringSubstringsMatch	substrings	Dopasowanie fragmentu
caseIgnoreMatch	equality	Równość łańcuchów
caseIgnoreOrderingMatch	ordering	Kolejność sortowania
caseIgnoreSubstringsMatch	substrings	Dopasowanie fragmentu

Serwer LDAP przetwarza obiekty przechowywane w bazie informacji katalogowej na różne sposoby. Posługuje się do tego celu zdefiniowanymi w schematach operatorami. Istnieją trzy klasy operatorów: operatory porównania, porządkowania i przeszukiwania. Porównanie ma na celu wykazanie, że dwa atrybuty mają dokładnie tą samą wartość.

Operator porządkowania jest wykorzystywany np. do sortowania wyników przeszukiwania. W zależności od atrybutu można zastosować różne operatory porządkowania, np. takie, które rozróżniają wielkie i małe litery bądź nie.

Operator przeszukiwania służy do wyłuskania z wartości atrybutu jakiegoś fragmentu. Tu również istotna może być np. kwestia rozróżniania wielkich i małych liter.

Operatory również mają przypisane im niskopoziomowe identyfikatory OID.

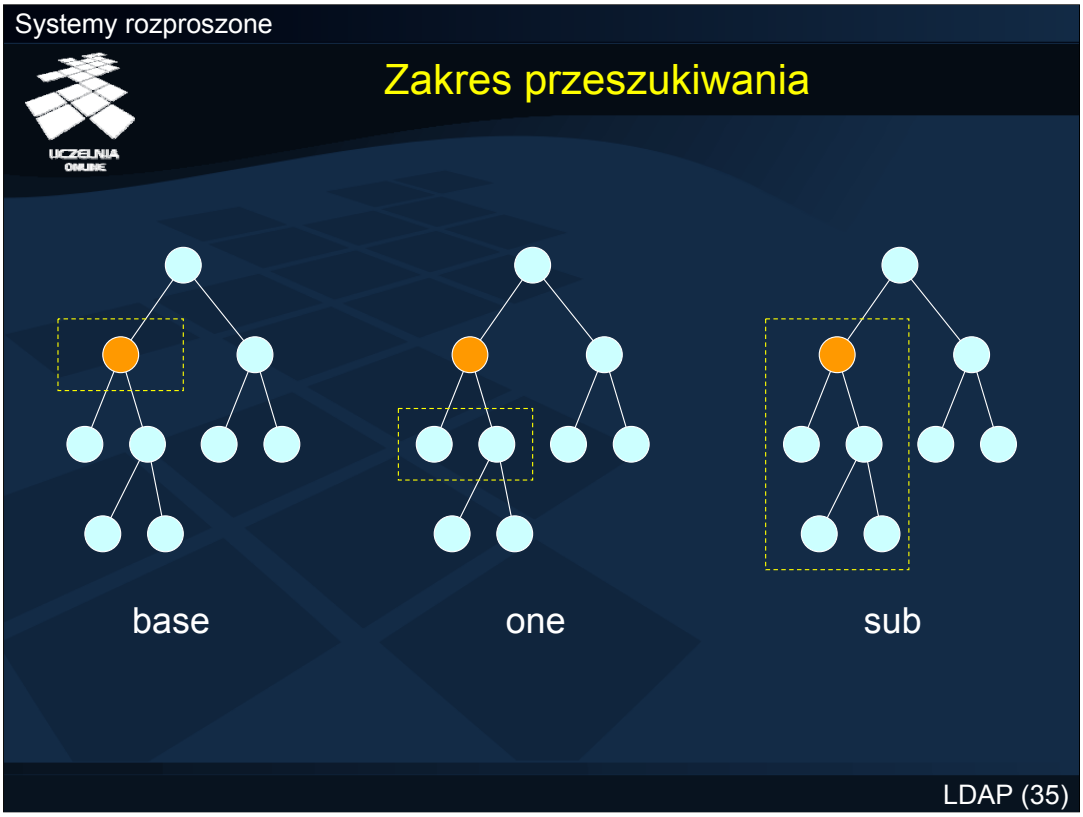


Jedną z najczęściej wykonywanych przez serwery LDAP operacji jest operacja przeszukiwania. Operacja ta wymaga wyspecyfikowania trzech parametrów: bazy, zakresu przeszukiwania i filtru.

Baza wskazuje na miejsce od którego należy rozpocząć przeszukiwanie drzewa. Dla zwiększenia efektywności realizacji zapytań należy wykorzystywać maksimum wiedzy na temat organizacji drzewa i wskazywać na miejsce rozpoczęcia przeszukiwania, które podejrzewamy, że jest najbliższe faktycznej lokalizacji poszukiwanego obiektu. Baza jest wskazywana poprzez nazwę wyróżnioną węzła startowego.

Zakres przeszukiwania określa jak głęboko należy sięgać do drzewa katalogowego w stosunku do bazy (zobacz nast. slajd).

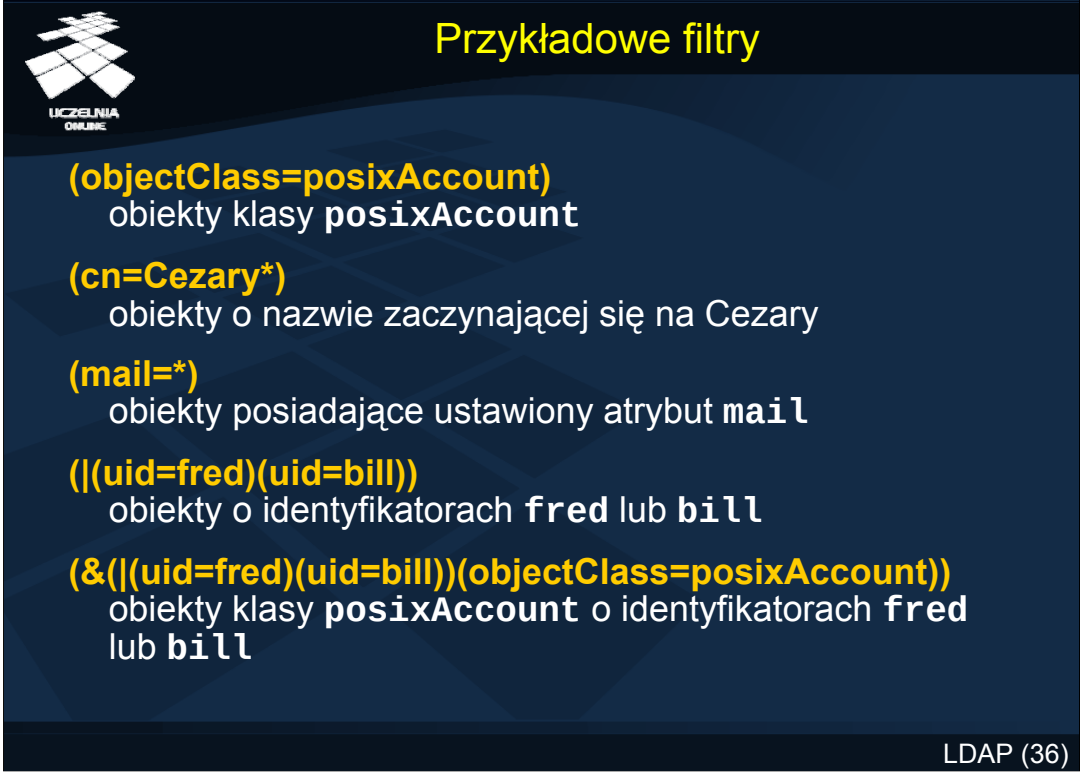
Filtr określa jakie warunki muszą spełniać poszczególne atrybuty przeglądanych obiektów, aby mogły być zaakceptowane jako wyniki poszukiwania. Do zapisu filtru stosuje się notację prefiksową z wykorzystaniem standardowych operatorów logicznych i operatorów porównania. Można wykorzystać też znaki specjalne * i ? reprezentujące odpowiednio dowolny ciąg znaków i dowolny pojedynczy znak.



Zakres przeszukiwania (ang. *scope*) może być określony jako *base*, *one* lub *sub*. Zostało to zobrazowane na rysunku. Przeszukiwanie typu *base* oznacza, że będzie odwiedzany tylko jeden węzeł, ten który został wskazany jako baza w zapytaniu. Przeszukiwanie takie ma sens np. w przypadku atrybutów wielowartościowych, bądź w celu sprawdzenia występowania atrybutu o określonej wartości.

Przeszukiwanie typu *one* oznacza, że odwiedzane będą tylko węzły o jeden poziom niższe od bazy. Przykładem zastosowania może być poszukiwanie węzła reprezentującego użytkownika w przypadku, gdy wiadomo, że wszyscy użytkownicy są zarejestrowani jako obiekty potomne w stosunku do jednego węzła nadrzędnego (bazy przeszukiwania).

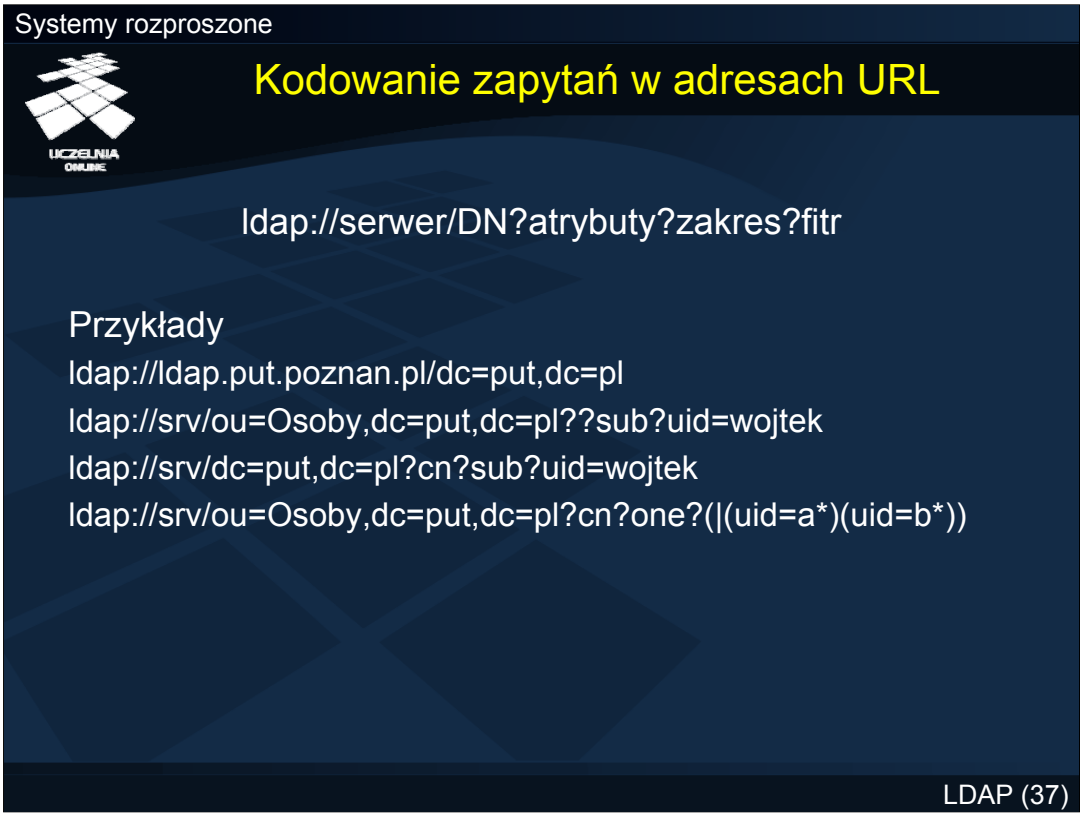
Najbardziej kosztownym przeszukiwaniem jest oczywiście przeszukiwanie typu *sub*, wymagające rekurencyjnego przeglądania całego poddrzewa. W tej metodzie ważne jest szczególnie precyzyjne wskazanie bazy przeszukiwania w celu uniknięcia przeglądania dużej części drzewa.



Slajd pokazuje przykładowe filtry ograniczające wyniki przeszukiwania. Pierwszy filtr wyszukuje wszystkie obiekty klasy *posixAccount*. Kolejny korzysta ze znaku specjalnego i wyszukuje obiekty (w tym przypadku użytkowników) o nazwie (imieniu/nazwisku) zaczynającym się na „Cezary”. Trzeci przykład przyrównuje wartość atrybutu *mail* do dowolnego ciągu znaków, a więc sama wartość atrybutu jest w tym przypadku nieistotna. Istotne jest natomiast to, że atrybut *mail* musi być zdefiniowany.

Czwarty przykład pokazuje jak zastosować alternatywę w zapytaniu. Sprawdzeniu podlega wartość atrybutu *uid*, która ma być równa *fred* lub *bill*. Operator „lub” znajduje się na początku (notacja prefiksowa).

Ostatni przykład jest rozwinięciem poprzedniego. Atrybut *uid* musi spełniać te warunki co poprzednio, a dodatkowo obiekt musi być wystąpieniem klasy *posixAccount*.



Zapytania do serwerów LDAP można kodować w postaci adresów URL zgodnie ze schematem przedstawionym na slajdzie. Pewne składowe zapytania można pomijać, przyjmują wtedy wartości domyślne. Np. jeżeli pominięta zostanie lista atrybutów, które mają być wynikiem zapytania, to zwrócone zostaną wszystkie. Domyślnym zakresem przeszukiwania jest *sub*. Domyślny filtr zwraca wszystkie obiekty.

Pierwszy przykład powinien zwrócić wszystkie obiekty z serwera *ldap.put.poznan.pl* począwszy od węzła *dc=put,dc=pl*.

Drugi przykład zwróci wszystkie atrybuty wszystkich węzły o identyfikatorze *uid=wojtek* dostępne na serwerze *srv* poniżej węzła *ou=Osoby,dc=put,dc=pl*.

Trzeci przykład zwróci wartości atrybutu *cn* dla wszystkich obiektów o identyfikatorze *uid=wojtek*.

W ostatnim przykładzie wynikiem przeszukiwania będą wartości atrybutu *cn* węzłów potomnych węzła *ou=Osoby,dc=put,dc=pl*, o ile atrybut *uid* tych obiektów ma wartość zaczynającą się na a lub b.

LDAP v3

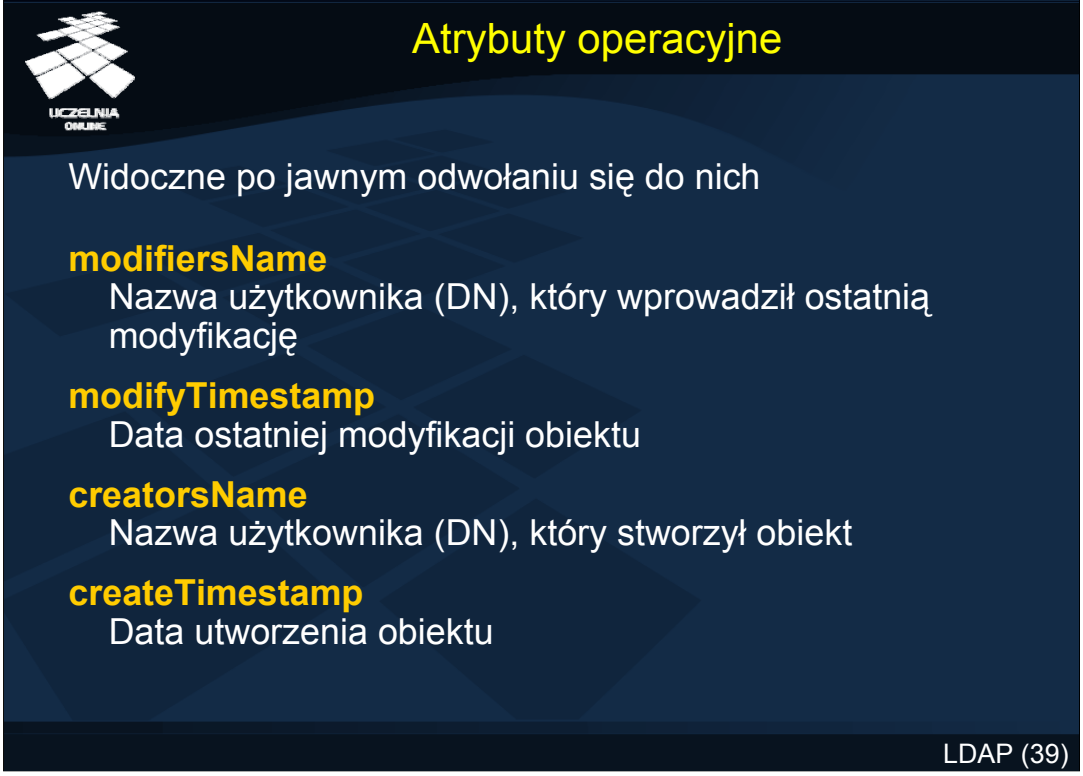
- Wsparcie dla języków narodowych (UTF-8)
- Odwołania do serwerów zewnętrznych (*referrals*)
- Bezpieczeństwo
  - uwierzytelnianie v2: anonymous, simple, Kerberos v4
  - v3: SASL (Simple Authentication and Security Layer)
- Rozszerzalność (extended operation), np. StartTLS
- Obsługa schematów danych (kontrola poprawności danych + podstawowe definicje)
- Introspekcja funkcjonalności serwera
- Atrybuty operacyjne

Protokół LDAP powstał to po, aby uprościć dostęp do usług katalogowych. Uproszczenie to poszło jednak tak daleko, że utracono część funkcjonalności oryginalnego protokołu DAP, która była bardzo przydatna. Stąd powstawały kolejne wydania tego protokołu: LDAP v2 i LDAP v3, które rozbudowywały początkową wersję, zwiększając tym samym stopień złożoności protokołu. Aktualnie wykorzystywane są wersje v2 i v3.

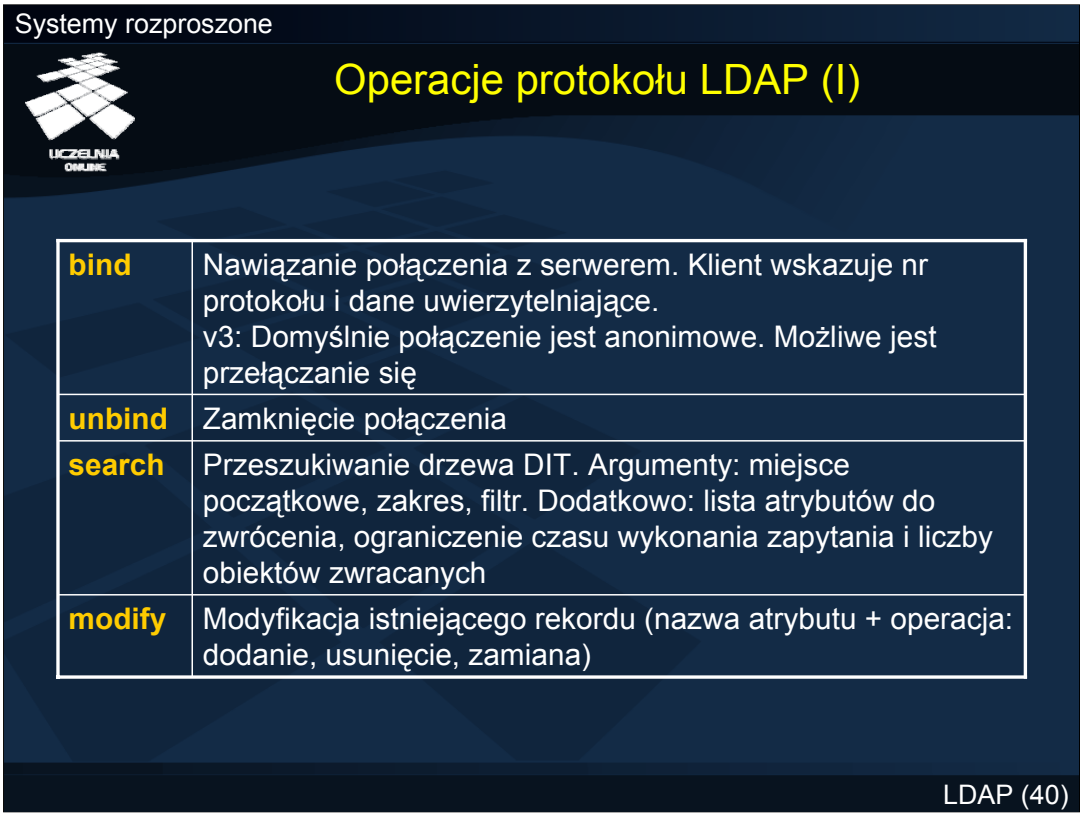
Wersja trzecia protokołu LDAP wprowadza kilka istotnych nowych możliwości. Jedną z nich jest wsparcie dla języków narodowych poprzez konsekwentne stosowanie kodowania UTF-8 do reprezentacji napisów tekstowych. W tej wersji pojawił się również mechanizm odwołań do serwerów zewnętrznych (ang. *referrals*). Poprawiono również kwestie związane z bezpieczeństwem: wersja druga umożliwiała jedynie uwierzytelnienie poprzez podawanie jawnego hasła lub poprzez system Kerberos. W wersji 3 obsługiwany jest bardzo popularny standard SASL. Związana z bezpieczeństwem jest również kwestia kodowania transmisji. Dodano obsługę komendy StartTLS włączającej dynamicznie mechanizm szyfrowania (ang. *Transport Layer Security*).

Omawiane schematy danych to również cecha wersji v3. W poprzednich wersjach możliwe było wprowadzanie danych niezgodnych ze schematem.

W bieżącej wersji można również posługując się protokołem LDAP uzyskać informacje o samym serwerze (oprogramowaniu, konfiguracji). Atrybuty operacyjne z kolei mogą posłużyć do pozyskiwania metainformacji o przechowywanych obiektach (zobacz nast. slajd).



Atrybuty operacyjne są to atrybuty obiektów, które domyślnie nie są widoczne, ale można je odczytać po jawnym wskazaniu. Służą do przechowywania metainformacji o obiektach drzewa katalogowego, związanych z datą utworzenia czy modyfikacji obiektu oraz użytkowniku, który dokonał wprowadzenia lub modyfikacji obiektu.

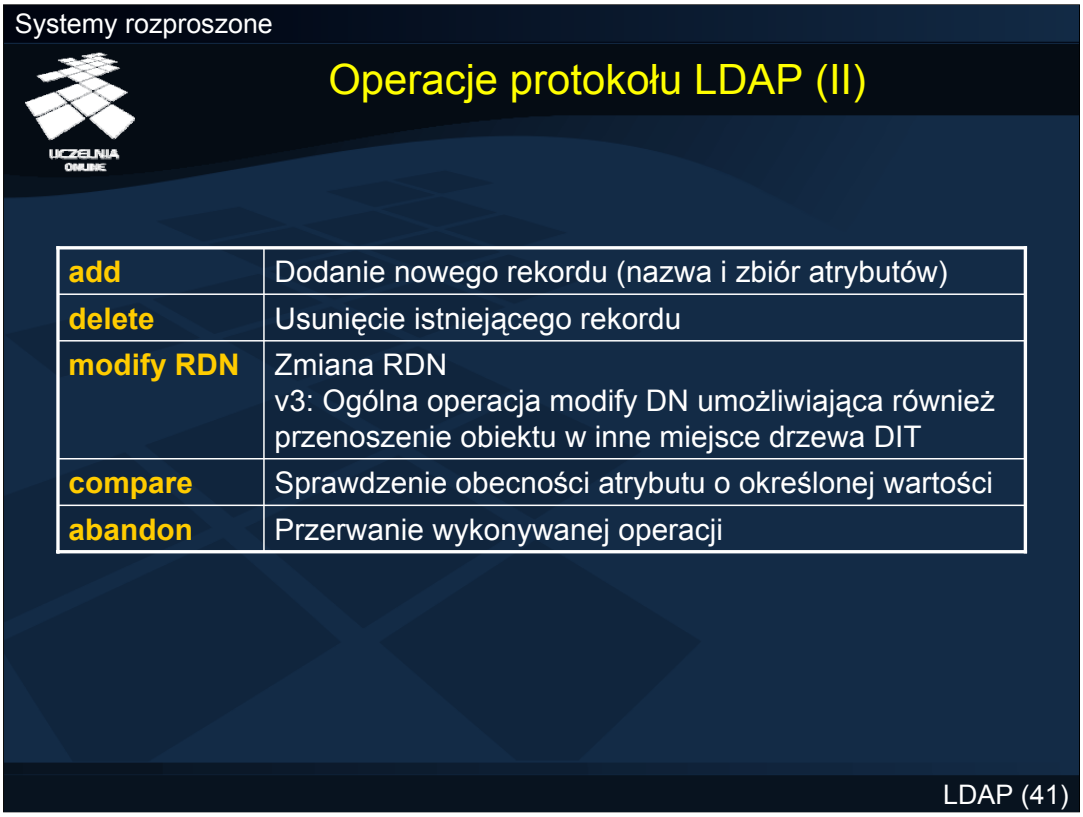


Protokół LDAP jest stosunkowo prostym protokołem (takim miał być z założenia). Ten i następny slajd wymieniają wszystkie operacje przewidziane w protokole.

Po nawiązaniu połączenia z serwerem użytkownik ma prawa użytkownika anonimowego. Zmianę uprawnień umożliwia komenda *bind*.

Komenda *search* realizuje przeszukiwanie drzewa. W zależności od konfiguracji serwer może nie przekazać wszystkich wyników z powodu przekroczenia dozwolonej maksymalnej liczby zwracanych obiektów lub z powodu przekroczenia czasu realizacji poszukiwań.

Komenda *modify* służy do modyfikacji dowolnego atrybutu (oprócz *dn*) istniejącego obiektu. Jako parametr wejściowy dla metody należy przekazać nazwę atrybutu oraz operację do wykonania: dodanie nowej wartości, usunięcie starej, bądź zamiana.

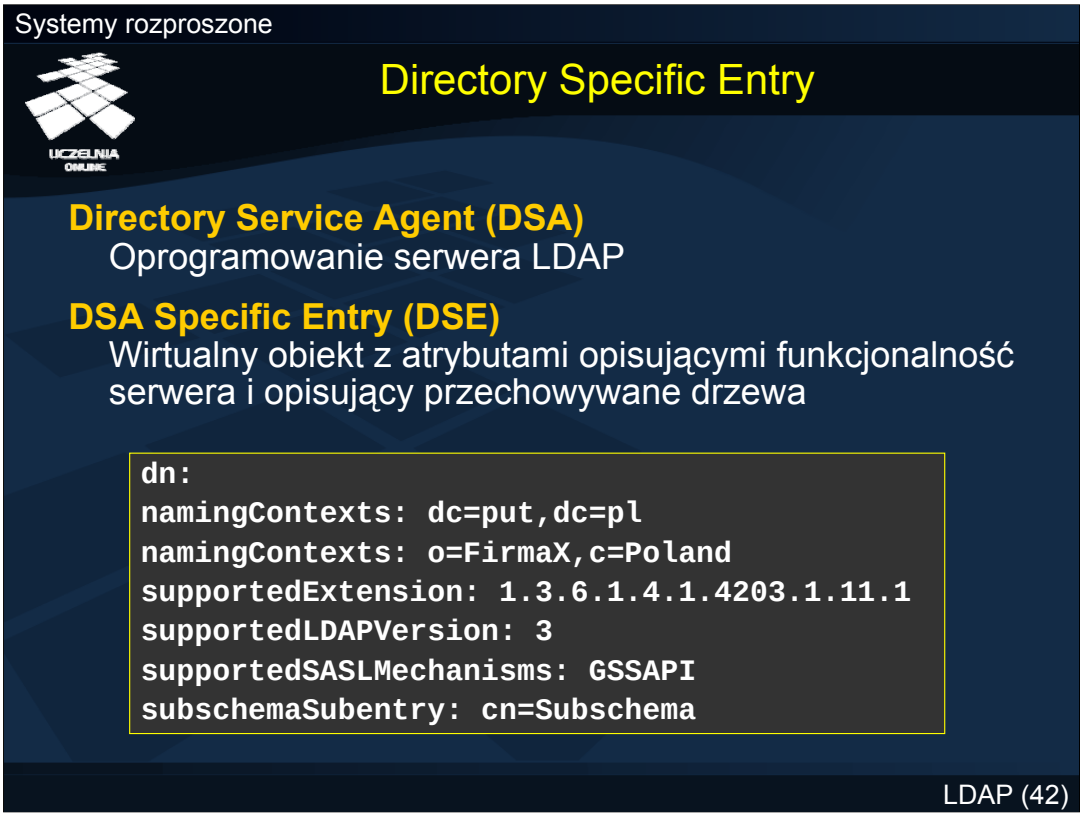


Komenda *add* służy do dodawania nowych obiektów. Należy podać nazwę wyróżnioną obiektu i pełną listę jego atrybutów.

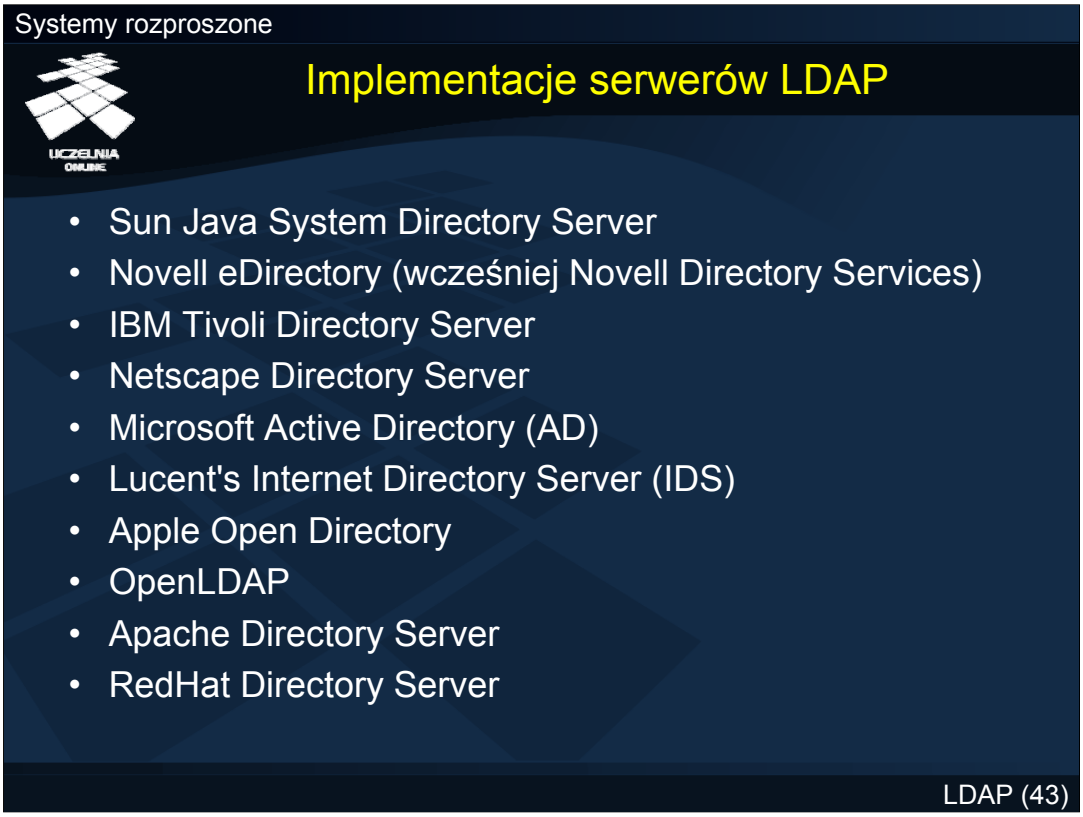
Komenda *modify RDN* służy do modyfikacji względnej nazwy wyróżnionej obiektu (czyli w ramach kontekstu). W wersji v3 protokołu LDAP wprowadzono również ogólną metodę *modify DN*, która pozwala na dowolną zmianę nazwy wyróżnionej, być może powodując tym samym przeniesienie obiektu w inne miejsce drzewa katalogowego.

Komenda *compare* służy do weryfikacji obecności atrybutu o określonej wartości.

Dowolną operację w trakcie jej trwania można przerwać komendą *abandon*. Jest ona przydatna, gdy zlecimy serwerowi współbieżne wykonywanie wielu operacji (np. czasochłonnego przeszukiwania).

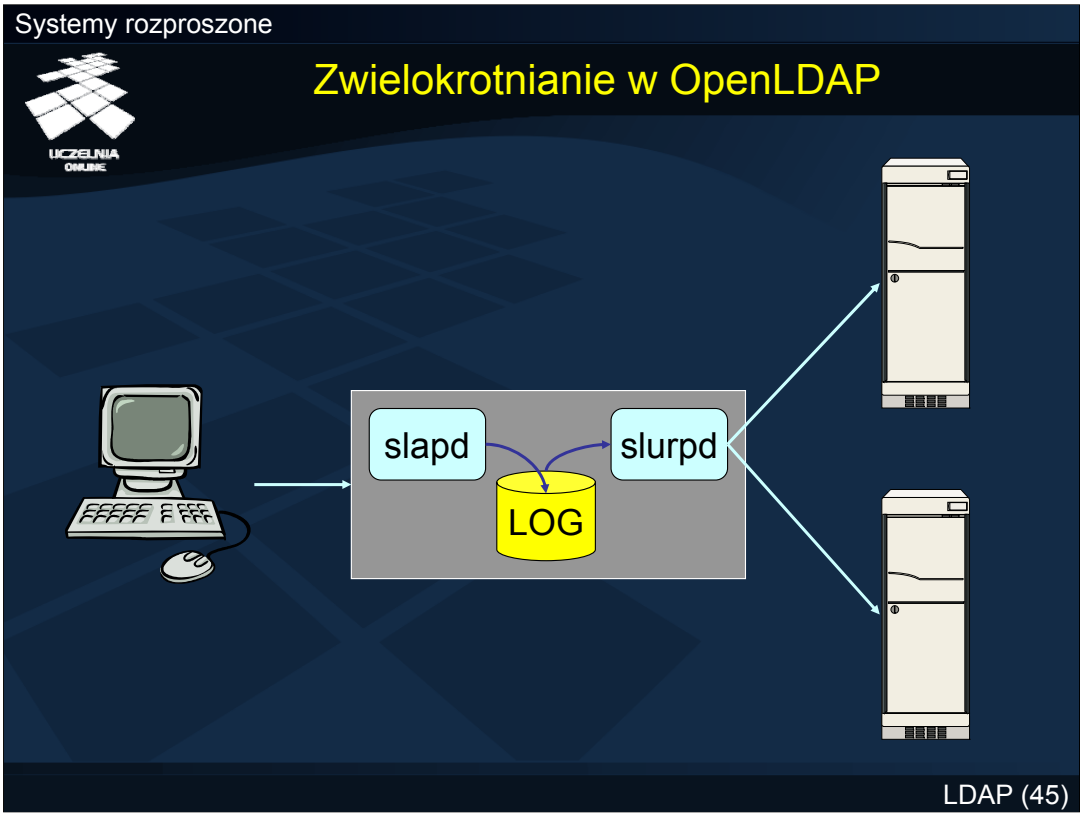


Skrót DSA oznacza oprogramowanie serwera implementujące protokół LDAP. W drzewie informacji katalogowej znajduje się specjalny węzeł określany jako DSE, który pozwala uzyskać metainformację dotyczącą samego serwera. Wśród informacji zawartych w DSE można znaleźć m.in.: nazwy głównych węzłów przechowywanych drzew katalogowych, wspierane wersje protokołu LDAP, wspierane mechanizmy uwierzytelniania, itp.

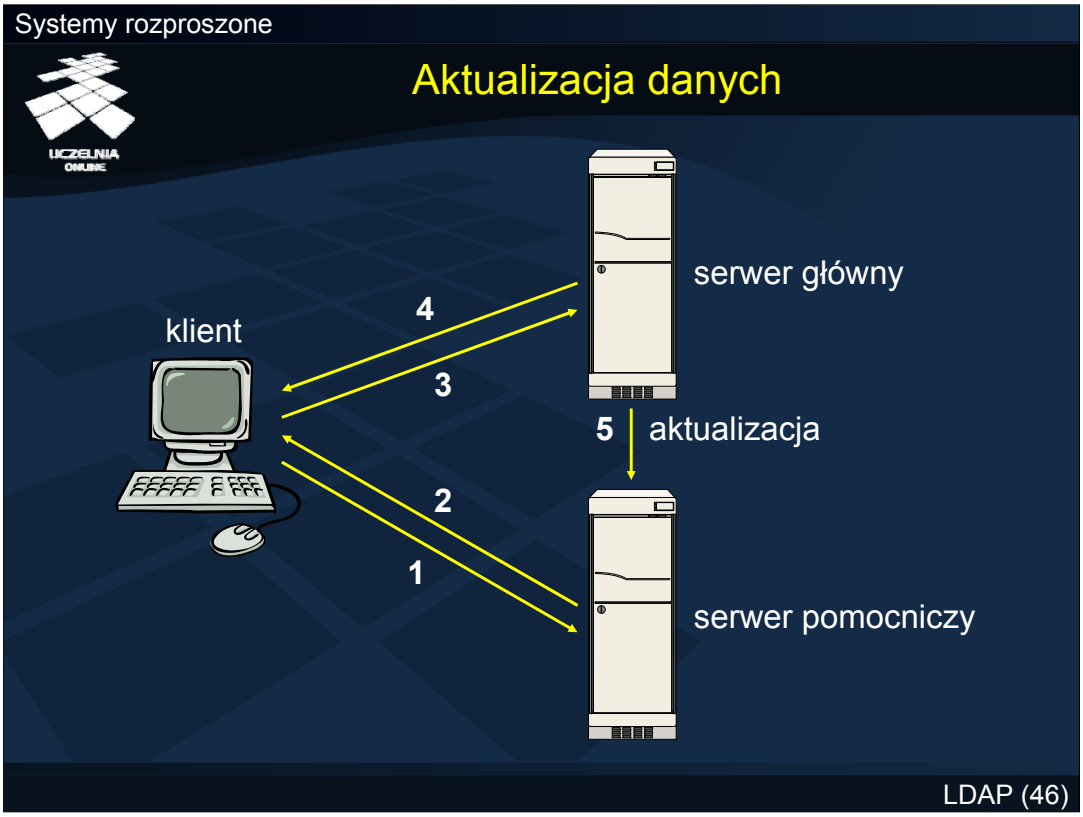


Istnieje wiele niezależnych implementacji protokołu LDAP, zarówno komercyjnych jak i typu *Open Source*. Duże doświadczenie w zastosowaniach usług katalogowych ma firma Novell, która oferowała usługę *Novell Directory Services — NDS* w swoim systemie operacyjnym NetWare. W nowszych wydaniach systemów Windows również występuje usługa katalogowa częściowo kompatybilna z protokołem LDAP o nazwie Active Directory. Ćwiczenia laboratoryjne zakładają wykorzystanie darmowego serwera OpenLDAP.

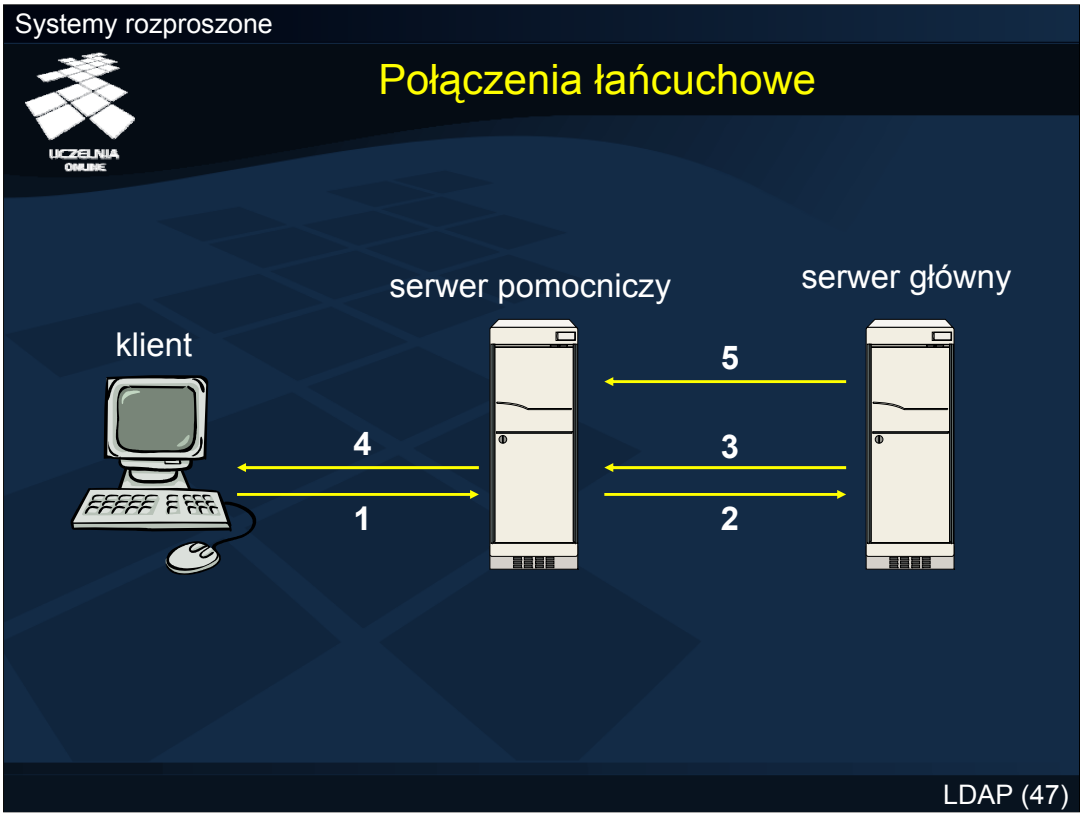
Kilka kolejnych slajdów prezentuje pokrótce cechy otwartego serwera OpenLDAP, który wykorzystywany będzie w ramach ćwiczeń laboratoryjnych. Serwer OpenLDAP obsługuje protokół LDAP w wersji 3. Jest to wielowątkowa implementacja oferująca możliwość pracy w środowisku zwielokrotnionym. Dane można składować za pośrednictwem serwera OpenLDAP w różnych formatach, w tym również odwołując się do relacyjnej bazy danych. Zaimplementowano w serwerze mechanizmy kontroli dostępu, umożliwiające precyzyjne określenie uprawnień poszczególnych użytkowników.



Serwer OpenLDAP umożliwia zwielokrotnianie danych przechowywanych w drzewie informacji katalogowej. W najprostszym trybie polega to na rejestrowaniu wszystkich zleceń modyfikujących napływających do głównego procesu serwera (*slapd*) i zapisywaniu ich do dziennika (log). Współbieżnie z procesem serwera pracuje proces kopiujący *slurpd*, którego zadaniem jest propagowanie wprowadzonych zmian w tej samej postaci do serwerów pomocniczych. Poprawne działanie mechanizmu zwielokrotniania wymaga wstępnego zainicjowania replik drzewa katalogowego do postaci identycznej jak oryginał. Propagacja zmian do serwerów pomocniczych nie musi być wykonywana natychmiast; może być wykonana w momencie gdy połączenie z serwerem pomocniczym jest możliwe.

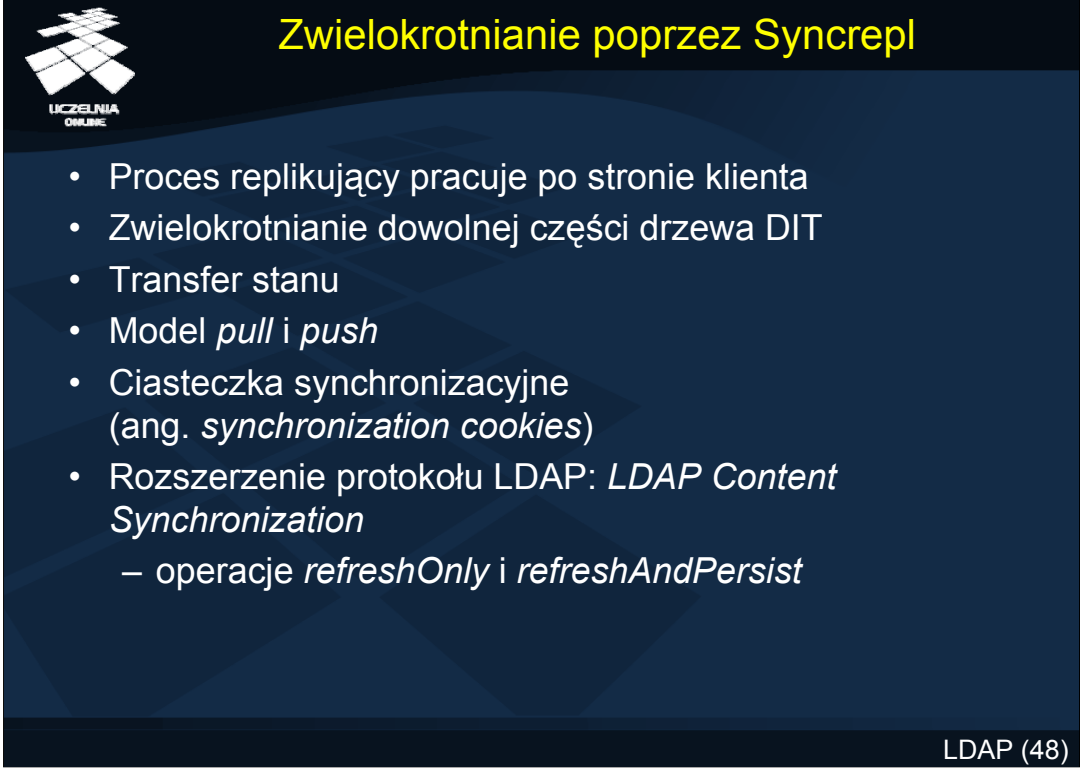


W podstawowym trybie zwielokrotniania serwery pomocnicze z założenia służą jedynie do udostępniania danych. Zlecenie modyfikujące klienta trafiając do serwera pomocniczego (1) może spowodować odesłanie wskazania na serwer, który obsługuje modyfikacje (2). Można do tego celu wykorzystać mechanizm odwołań (ang. *referrals*). Klient uzyskując nowy adres kieruje zlecenie modyfikujące do serwera głównego (3) i uzyskuje potwierdzenie wykonania operacji (4). W międzyczasie do serwera pomocniczego wysyłany jest komunikat aktualizacyjny (5).



Aktualizacja danych poprzez serwer pomocniczy może być zautomatyzowana. Na slajdzie pokazano wykorzystanie połączenia łańcuchowego, w którym serwer pomocniczy – w przypadku zleceń modyfikujących – pełni rolę serwera pośredniczącego i przekazuje operację do wykonania serwerowi głównemu (2). Ten po wykonaniu modyfikacji odsyła potwierdzenie (3), które dociera ostatecznie do klienta (4). Serwer główny w międzyczasie dokonuje aktualizacji serwera pomocniczego (5).

Istotną zaletą przedstawionego schematu działania jest ukrycie przed klientem faktu zwielokrotnienia serwera. Serwer pomocniczy zachowuje się z punktu widzenia klienta dokładnie tak samo jak serwer główny.



Drugim podejściem stosowanym w OpenLDAP do zwielokrotniania jest rozszerzenie protokołu LDAP o dodatkowe operacje związane z odświeżaniem kopii. W tym podejściu obsługa zwielokrotniania realizowana jest poprzez proces pracujący po stronie klienta. Zaletą tego podejścia jest możliwość precyzyjnej konfiguracji które fragmenty drzewa katalogowego mają być synchronizowane z serwerem głównym. Obsługiwane są zarówno tryby pobierania (*pull*) jak i wypychania (*push*). Dla przechowania informacji o stanie serwerów stosuje się tzw. ciasteczka synchronizacyjne określające moment czasowy ostatniej synchronizacji. W odróżnieniu od poprzedniej metody, gdzie aktualizacja sprowadzała się do powielenia sekwencji tych samych operacji modyfikujących na serwerach pomocniczy, w tym podejściu stosuje transfer stanu. Obiekty, które uległy zmianie przesyłane są w całości, bez względu na ilość modyfikacji jakie miały miejsce. Ponieważ obiekty w usługach katalogowych mają stosunkowo niewielką objętość, podejście to daje dobre rezultaty.

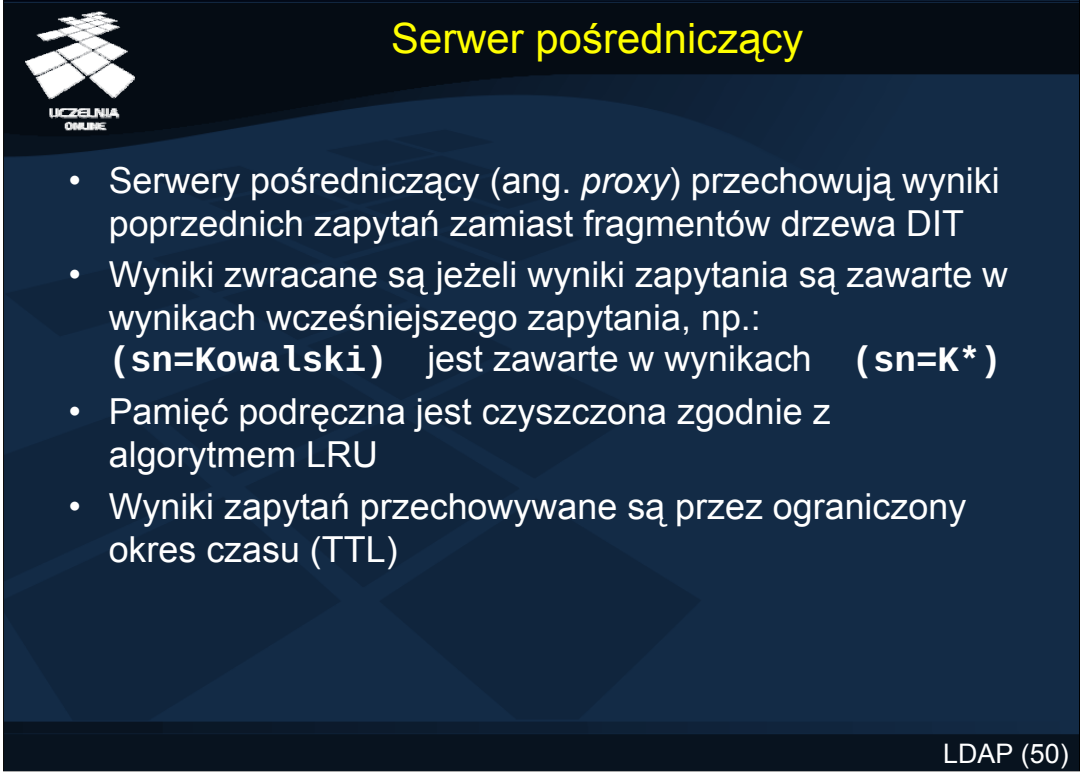
Syncrepl

- Aktualizacja *present phase*: całe zmodyfikowane obiekty + informacja o obiektach, które nadal są w drzewie
- Aktualizacja *delete phase*: całe zmodyfikowane obiekty + informacja o obiektach, które zostały usunięte — wymaga przechowywania stanu po stronie serwera głównego
- Identyfikacja obiektów poprzez atrybut *entryUUID* — *dn* może się zmienić
- Reprezentacja stanu *contextCSN* — etykieta czasowa ostatniej zmiany

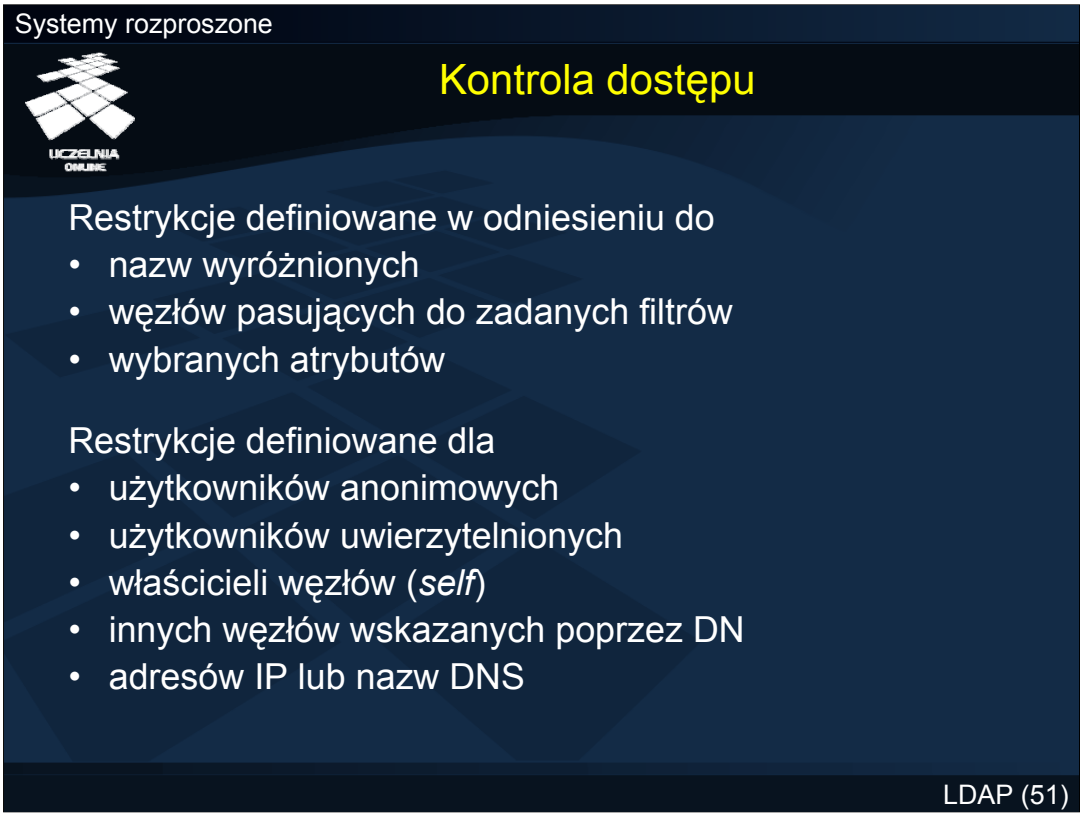
Synchronizacja pomiędzy serwerami przebiega na dwa sposoby. W pierwszej metodzie (*present phase*) przesyła się między serwerami całe obiekty, które zostały zmodyfikowane oraz informację o obiektach, które nadal są w drzewie katalogowym. Na podstawie tych danych serwer pomocniczy może zrekonstruować swoje drzewo tak, aby było identyczne z oryginałem. Rekonstrukcja polega na wprowadzeniu zmodyfikowanych obiektów (obejmuje to również wprowadzenie nowych obiektów), oraz na usunięciu obiektów z lokalnego drzewa, które nie są na liście przesłanej przez serwer główny.

Drugie podejście nie wymaga przesyłania informacji o obiektach, które się nie zmieniły. Jest to podejście *delete phase*, w którym przesyłane są całe zmodyfikowane obiekty (jak poprzednio) oraz informacja o obiektach, które zostały usunięte. Na serwerze głównym trzeba jednak w tym przypadku przechować dodatkową informację o usuniętych obiektach po to, by ją później przekazać serwerom pomocniczym. Zaletą tej metody są małe komunikaty w przypadku braku zmian.

Protokół *Syncrepl* wymaga jednoznacznej identyfikacji obiektów z drzewa katalogowego. Nazwy wyróżnione nie są w tym przypadku odpowiednie, ponieważ mogą się zmieniać w czasie. Z tego powodu protokół korzysta z atrybutów *entryUUID*, które są nadawane podczas tworzenia obiektów i nigdy się nie zmieniają.

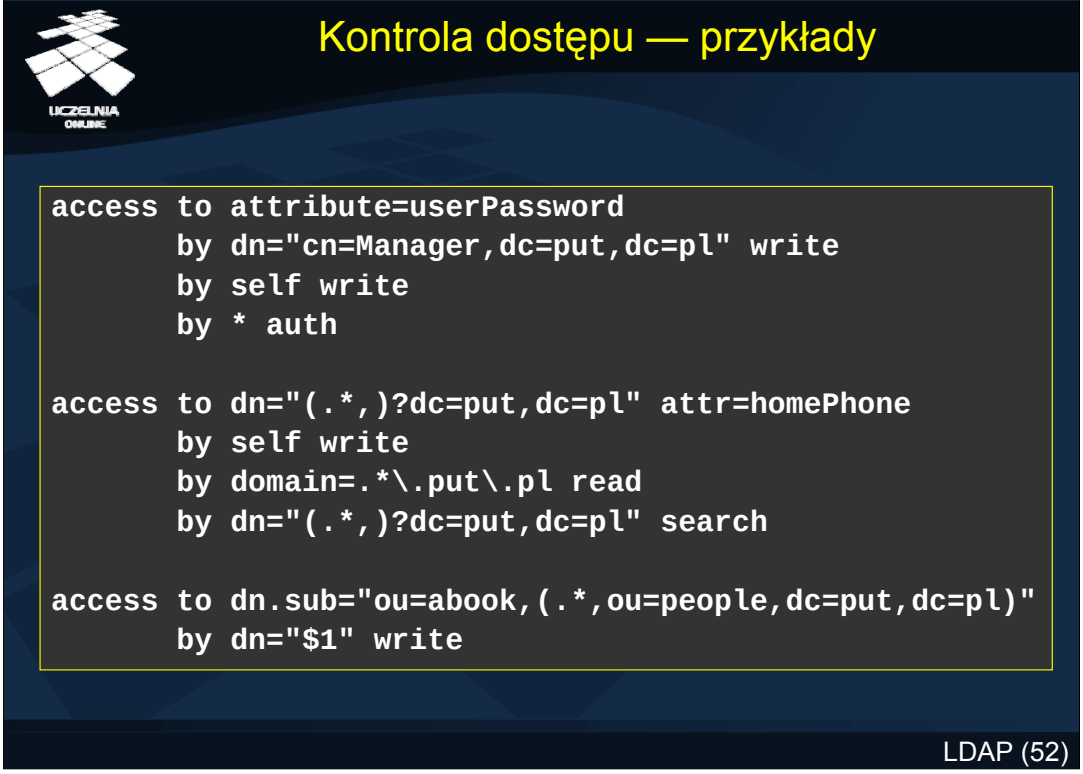


Serwer OpenLDAP może pracować serwer pośredniczący (ang. *proxy*). W tym trybie serwer nie przechowuje rekordów, a jedynie wyniki wcześniejszych zapytań z uwzględnieniem możliwości zawierania się wyników jednego zapytania w wynikach innego zapytania. Pamięć podręczna ma skończony rozmiar, więc jest czyszczona w miarę potrzeb zgodnie z algorytmem LRU (ang. *Least Recenty Used* – najdłużej niewykorzystywany). Istnieje też limit czasowy przechowywania danych w pamięci podręcznej.



Dane przechowywane na serwerach usług katalogowych bardzo często nie mogą być udostępniane wszystkim bez ograniczeń. Z tego powodu mechanizm kontroli dostępu jest niezbędny do poprawnego wdrożenia usługi. Mechanizm ten nie jest jednak objęty standardem, stąd każdy produkt może ten problem inaczej rozwiązywać.

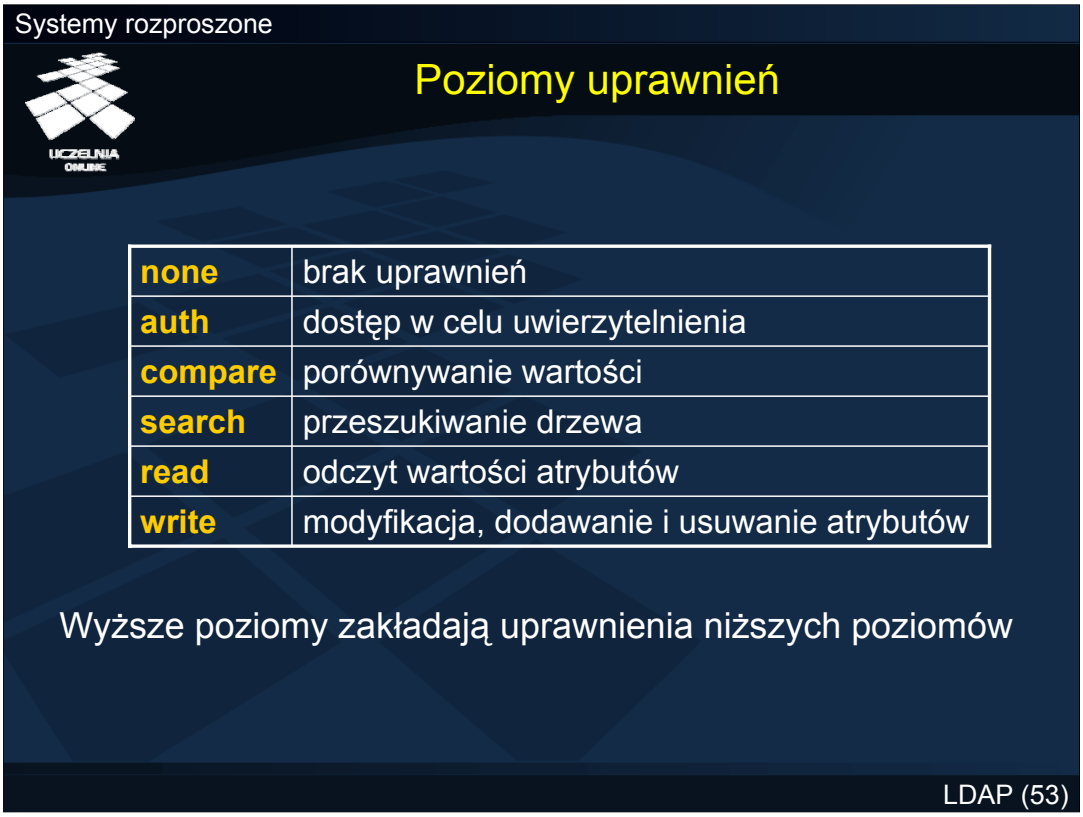
W serwerze OpenLDAP reguły dotyczące praw dostępu ustalane są w odniesieniu do węzłów o konkretnych nazwach wyróżnionych, do węzłów o nazwach pasujących do zadanych filtrów, i na niższym poziomie do konkretnych atrybutów. Adresatami restrykcji są użytkownicy anonimowi (niezalogowani), użytkownicy uwierzytelnieni, właściciele węzłów, użytkownicy identyfikowani poprzez nazwy wyróżnione i w końcu klienci posługujący się wskazanymi adresami IP czy nazwami domenowymi.



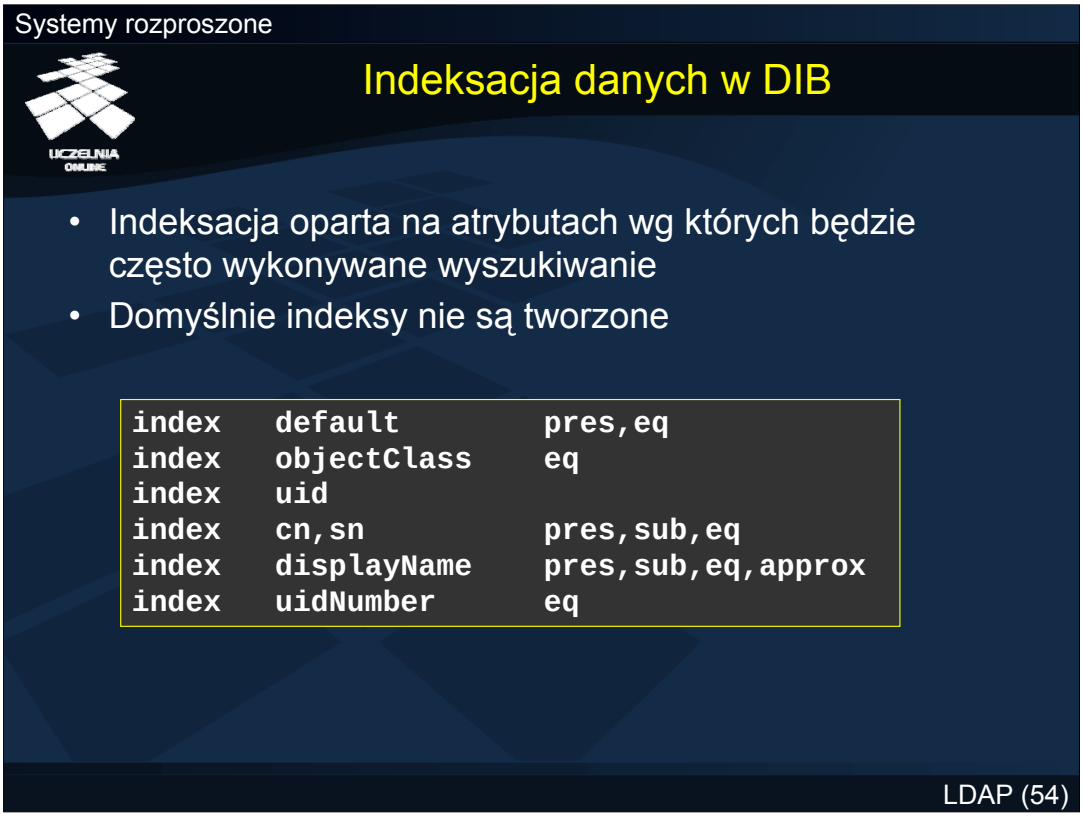
Slajd pokazuje trzy przykładowe reguły dostępu. Pierwsza reguła określa dostęp do atrybutów *userPassword* wszystkich obiektów. Ponieważ atrybut ten przechowuje hasło, dostęp do niego musi być mocno ograniczony. I tak: administrator systemu (użytkownik o dn=„cn=Manager,dc=put,dc=pl”) ma możliwość modyfikacji tego pola (prawo *write*), podobnie właściciel obiektu. Natomiast wszyscy pozostali mogą jedynie uwierzytelniać się w oparciu o to pole (prawo *auth*).

W drugiej regule wykorzystano wyrażenia regularne. Dostęp do wszystkich węzłów podrzędnych względem dc=put,dc=pl, do atrybutu *homePhone* jest możliwy dla: właściciela do modyfikacji, dla klientów z domeny *put.pl* do odczytu i dla pozostałych użytkowników identyfikowanych w drzewie katalogowym do przeszukiwania (prawo *search*).

Trzecia reguła zakłada, że użytkownicy mają zdefiniowane węzły podrzędne *ou=abook* (książka adresowa). Reguła ta daje możliwość modyfikacji zawartości tego węzła dla użytkownika, który jest węzłem nadrzędnym w stosunku do *ou=abook*. Nazwa $1 odnosi się w tym przypadku do fragmentu nazwy wyróżnionej znajdującej się linijkę wyżej w nawiasach.

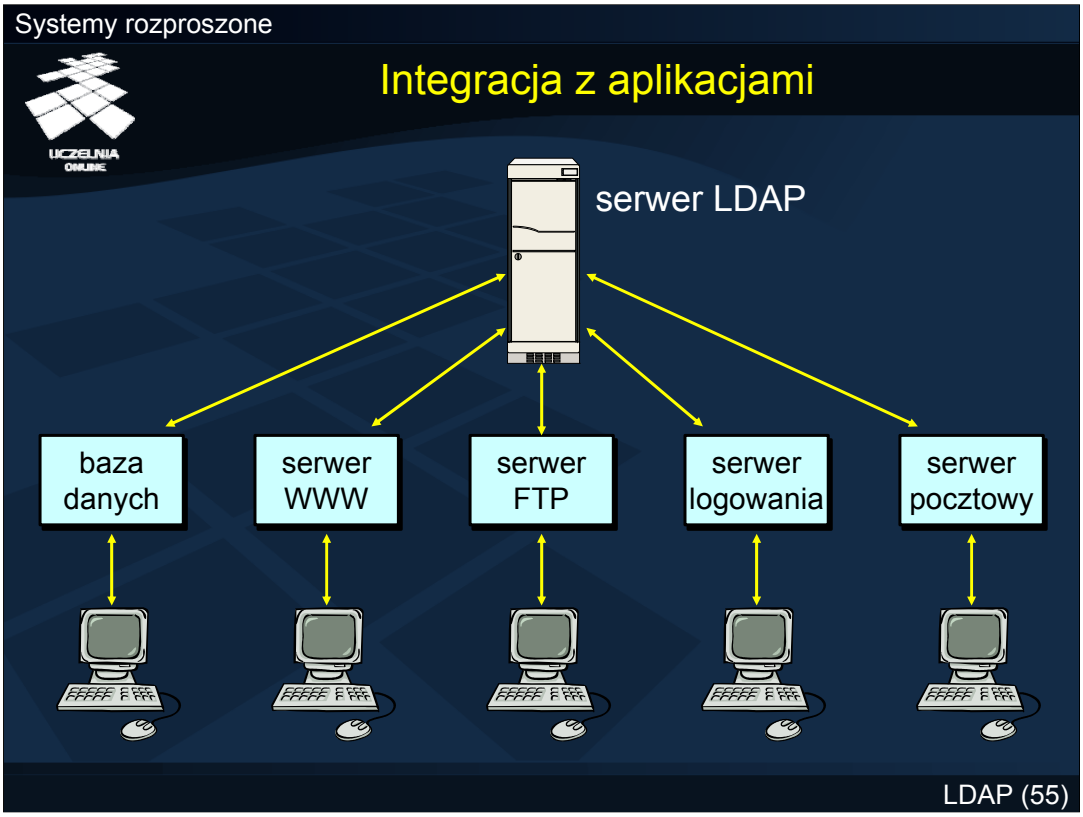


W serwerze OpenLDAP zdefiniowano wymienione w tabeli poziomy uprawnień. Prawo do modyfikacji (*write*) zezwala na wykonywanie wszystkich operacji. Każdy kolejny poziom redukuje to prawo, aż do poziomu *auth*, które zezwala jedynie na przeprowadzenie uwierzytelniania (podczas wykonywania operacji *bind*). Prawo *search* jest nieco silniejsze od *compare* gdyż umożliwia automatyczne wyszukiwanie informacji w oparciu o zadane kryterium. Prawo *compare* pozwala jedynie sprawdzić pojedynczy przypadek.

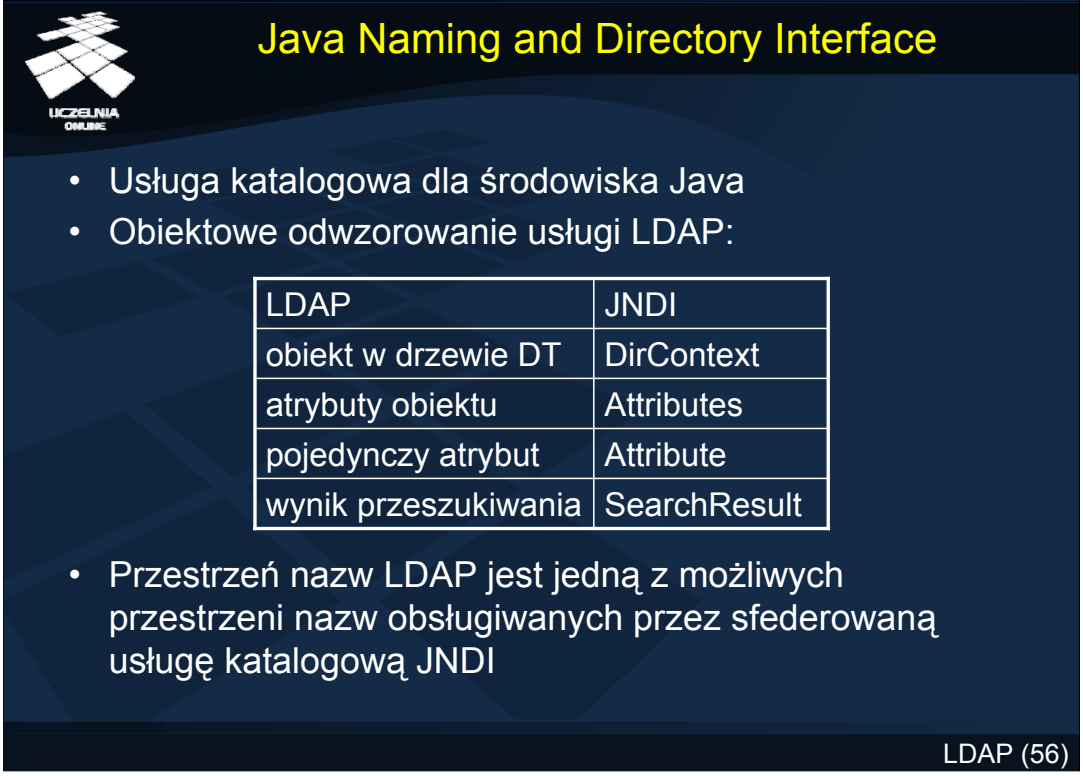


Wyszukiwanie danych w drzewie katalogowym można optymalizować poprzez założenie indeksów (podobnie jak w bazach danych). Indeksy należy założyć na tych atrybutach wg których najczęściej realizowane są operacje wyszukiwania informacji. Z pewnością będą to atrybuty *objectClass* czy *uid* w przypadku zarządzania bazą danych użytkowników.

Powyższy fragment konfiguracji serwer OpenLDAP aktywuje indeksy dla wymienionych atrybutów. Indeksy mogą mieć postać testu obecności (*pres*), równości (*eq*) i testu zawierania (*sub*) czy przybliżonego zawierania (*approx*).

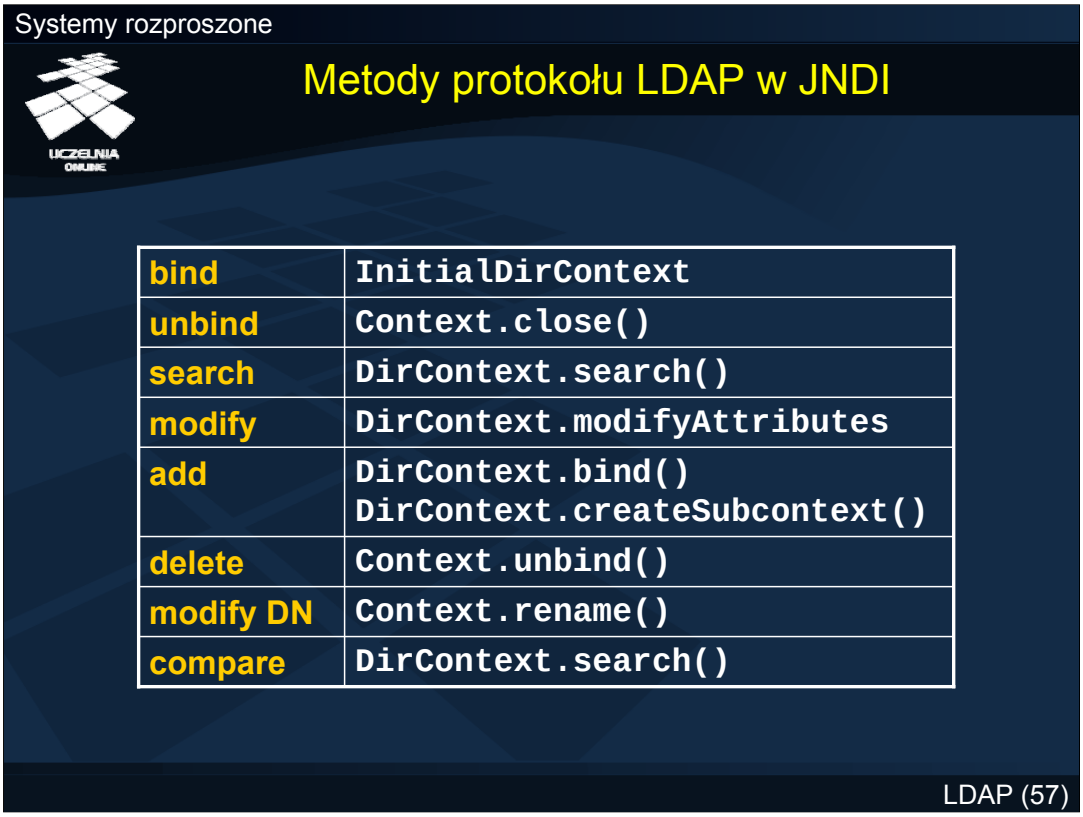


Uruchomienie serwera LDAP pozwala na późniejszą integrację wielu innych usług z konfiguracją zawartą w katalogu. Zamiast więc utrzymywać np. listy użytkowników niezależnie dla każdej z usług typu WWW, FTP, baza danych, serwer pocztowy – można po prostu zintegrować je z działającym serwerem LDAP. Jest to bardzo wygodne rozwiązanie dla administratora, ponieważ umożliwia mu dostosowywanie konfiguracji z poziomu jednej, standardowej usługi.



Interfejsy programowe, jak to zostało już powiedziane, nie są objęte standardem usługi X.500. Dla każdego produktu i dla każdego języka programowania odpowiednie API może więc wyglądać inaczej. Niniejszy slajd prezentuje pewne założenia odnośnie przykładowego interfejsu dla języka Java. Interfejs ten jest częścią większego projektu *Java Naming and Directory Interface*, który jest próbą szerszego spojrzenia do usługi katalogowe. Protokół LDAP i przestrzeń danych obsługiwanych przez ten protokół są tu traktowane jako jedna z możliwości.

Ze względu na charakterystykę języka Java (pełną obiektowość) interfejs programistyczny jest zorientowany obiektowo. Poszczególne elementy usługi katalogowej, takie jak: węzły, konteksty, atrybuty, są tu reprezentowane przez odpowiednie obiekty.



Podejście obiektowe widać również w implementacji metod protokołu LDAP. Są one realizowane jako metody klas reprezentujących odpowiednie abstrakcje usługi katalogowej. Tabelka pokazuje mapowanie metod protokołu LDAP na metody klas interfejsu JNDI.

Stosowanie interfejsu obiektowego przy dostępie do usługi katalogowej jest wygodne, ponieważ dobrze integruje się z samą aplikacją, która najprawdopodobniej również jest implementowana obiektowo. Wywołania metod wyszukujących informację w drzewie czy ją modyfikujących są naturalne i łatwe do opanowania.

Dokumentacja

RFC 1777	LDAPv2
RFC 2251–2256	Podstawowe dokumenty opisujące LDAPv3
RFC 2829	Metody autoryzacji protokołu LDAP
RFC 2830	LDAP i TLS (szyfrowanie)
RFC 3377	Specyfikacja techniczna LDAPv3

Ważne adresy

http://www.openldap.org — serwer OpenLDAP

http://www.padl.com — pakiety **`pam_ldap`**, **`nss_ldap`**

Więcej informacji o protokole LDAP można znaleźć w odpowiednich dokumentach RFC. Serwer OpenLDAP i jego dokumentację można pobrać z serwisu www.openldap.org. Pod adresem www.padl.com można z kolei znaleźć pakiety programowe umożliwiające systemową integrację informacji o użytkownikach zarejestrowanych na serwerze LDAP w systemie Unix.