VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NÁSTROJ PRO DOTAZOVÁNÍ SSSD DATABÁZE
TOOL FOR QUERYING SSSD DATABASE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE DAVID BAMBUŠEK
AUTHOR

VEDOUCÍ PRÁCE Doc. Dr. Ing. DUŠAN KOLÁŘ
SUPERVISOR

BRNO 2013

Abstrakt

Práce je zaměřena na databáze a konkrétně pak na databází SSSD. SSSD je služba, která poskytuje jedno kompletní rozhraní pro přístup k různým vzdáleným poskytovatelům a ověřovatelům identit spolu s možností informace z nich získané ukládat v lokální cache k offline použití. Práce se zabývá jak databázemi obecně, tak pak hlavně LDAP a LDB, které jsou použity v SSSD. Dále popsuje architekturu a samotnou funkci SSSD. Hlavním cílem pak bylo vytvořit aplikaci, která bude administrátorům poskytovat možnost prohlížet všechna data uložená v databázi SSSD.

Abstract

This thesis is focused on databases, concretely on SSSD database. SSSD is a set of daemons providing an option to access various identity and authentication resources through one simple application, that also offers offline caching. Thesis descrbes general information about databases, but mainly focuses on LDAP and LDB, that are used in SSSD. In addition also describes function and architecture of SSSD. Min goal of this thesis was to create a tool, that will be able to query all the data stored in SSSD database.

Klíčová slova

Databáze, LDAP, LDB, SSSD, Red Hat, dotazovací nástroj

Keywords

Databases, LDAP, LDB, SSSD, Red Hat, querying tool

Citace

David Bambušek: Tool for querying SSSD database, bakalářská práce, Brno, FIT VUT v Brně, 2013

Tool for querying SSSD database

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Doc. Dr. Ing. Dušana Koláře

. . . .
David Bambušek
April 23, 2013

Poděkování

Velice rád bych poděkoval vedoucímu mé bakalářské práce Doc. Dr. Ing. Dušanovi Kolářovi, který mi poskytl cenné pedagogické a věcné rady k vypracování mé práce. Stejně velký dík patří odbornému konzultantovi Ing. Janu Zelenému, za všechny potřebné rady a informace, které mi pomohly k úspěšnému vypracování mé práce. Na místě je také poděkovat firmě Red Hat, která mi umožnila vypracovávat práci pod její záštitou, stejně tak jako všem jejím zaměstancům, hlavně pak Ing. Jakubu Hrozkovi, kteří se podíleli na revizi mého kódu a poskytovali další věcné rady a doporučení.

Contents

Chapter 1

Introduction

Humanity has gone through a lot of eras since its dawning 200,000 years ago. As time passed by, we moved from stone age, passed bronze and iron age to get to modern age, sometimes called the silicon age as to refer to enormous boom of computers. And indeed, today we live in a world that could barely be functional without computers, no matter what you do and where you go, you are surrounded by them. Human society has also been changing during ages, from agricultural, passing industrial we got to stage where our society is knowledge-based. Thanks to the Internet, society has become one global connected network of people and the most important thing that makes people powerful is knowledge, in other words information. Unfortunately people are not able to store all the information they know in their own brains and therefore we are forced to use computers to help us with this task. For purposes of storing huge quantities of information that we need, we created computer databases.

Databases can store various types of data, reflecting real models from our world or can store purely abstract data. To identify ourselves on the Internet or any other networks, we use our virtual profiles, that store data about ourselves and about our membership in certain groups or companies. This thesis describes a tool, that is able to query such a database of users and additional information provided for managing virtual identity. Concretely it is a SSSD database. SSSD is a set of daemons providing an option to access various identity and authentication resources through one simple application, that also offers offline caching.

In chapter two, we will look generally on various existing types of databases and make more detailed description of the most used one, that is relational database.

Chapter three is focused on directory services - LDAP and LDB, which is not directory service by itself, but it is an abstract layer over lower key-value types of databases, offering LDAP-like API. LDB is the database used in SSSD.

In chapter four we will take a closer look on SSSD itself, we will introduce basic purpose, function and architecture and in chapter five a complete description of a SSSD database will follow.

The sixth chapter describes the tool for querying SSSD databases, there we will find a complete architecture of this application, its UI and examples of usage.

Conclusion and development will be stated in the last seventh chapter.

Chapter 2

Introduction to databases

[1] [5] Term database can be understood as a set or collection of data, that is somehow organized. It is typical, that a database reflects a real existing concept, system, structure or information, under which we can imagine for example cities and their populations, company employees or items in a store. We can work with this database and store, change, delete or query information stored in it.

To work with a database, we use a database management system (DBMS), what is a software allowing us to run all mentioned operations and to administrate the database. Most known and widely used DBMS are for sure MySQL, SQLITE, Microsoft Access or PostgreSQL.

When speaking about database, we usually understand this term as both data and their structure so as database management system, but formally „database“ is just data wired to its data structures.

We can divide function of DBMS into four groups:

1. Data control - maintaining integrity, taking care of security, monitoring, dealing with permissions to work with a database (creating and managing users)
2. Data definition - creating/modifying/removing data structures to/from a database
3. Data maintenance - inserting/updating/deleting data from a database
4. Data retrieval - data mining done by users to work with received information or to proceed it for other purposes by querying and analyzing

Each database with its DBMS works and looks accordingly to its database model. We will get back to each type in next subsections, for now to mention them, they are historically divided to three major groups - navigational, relational and post relational databases.

During first era in 1960's, there were two main representants of navigational model - hierarchical model developed at IBM and the Codastl(Network) model. Navigation in Codastyl was based on linked data creating huge network. It was possible to search for entries using their primary key (CALC key), by using relationships from one entry to another or by going through all entries sequentially. IBM's DBMS IMS was quite similar to Codastl, but instead of network model it used stricter hierarchical model.

In 1970's the world first encountered relational DBMS, which was introduced by Edgar Codd from IBM, more detailed description will be offered in next section. Main difference is that a database is formed from tables, each used for different entity. Main idea in terms of how to search for entries is that we should search data by content and not by following links as it was with navigational databases. Relational databases became the most used types of databases and have persisted on that spot until now and the dominant language used to work with them is SQL.

So far last era of databases, called post-relational and also known as NoSQL databases, showed up at the dawn of new millennium. These databases are document-oriented and offer quick key-value storage. They are mostly used in situations when we store large quantity of information and where the relationship between them is not that important. These databases found their place at social networks like Facebook or Twitter to store comments, tweets and other mass quantity information.

Languages used for communication with databases have each its own specific function:

- Data definition languages - define data types and their relationships
- Data manipulation languages - used for inserting/updating/deleting data
- Query languages - used for searching for data

Each database model has its own language. Most know and used languages are, already mentioned, SQL, then OQL used in object model databases or XQuery used with XML databases.

2.1 Relational database

2.1.1 Overview and terminology

A relational database is a set of data entries, that are organized as a collection of tables. Relational database is created upon a basis of relational model and a software to operate this database is called a relational database management system (RDBMS). It is nowadays the most used type of database, that can satisfy needs for solutions of most common problems and can be applied on most models that can be found in our world. The most simplified view on relational database is that it consists of tables, that are composed of rows, where each field represents one of attributes.

Relational database theory is a mathematical theory and it has its own terminology, these terms are slightly different from those terms, that we use when talking about SQL, which is the main language operating with relational databases. Table 2.1 shows the differences.

mathematical theory	SQL
relation, base relvar	table
tuple	row
attribute	column name
attribute value	column data
derived relvar	query result

Table 2.1: mathematical vs. SQL terminology

2.1.2 Relational model

The basic idea of relational model is that all data is mathematically represented as n-ary relations, which are subsets of a cartesian product of n domains. The view on data in this mathematical model is done by two-valued predicate logic, where each proposition can be either true or false (later there were attempts to change this to three or four valued predicate logic by adding unknown value and then valid an invalid unknown). Data are handled due to rules of relational algebra or a relational calculus.

Each relational database has constraints, which lead its designer to create a consistent representation of some information model. The process of creating consistent database is called normalization, which leads to selection of most suitable logically equivalent alternative of the database.

Basic building stone of a relational database is domain, usually referred as type. Tuple, that is ordered set of attributes. Attribute consists of attribute name and data type name. Every relation is composed from a heading and a body. Heading is a collection of attributes and body is the rest, meaning a set of n-tuples. Relation, which is visualized as a table, consists of a set of n-tuples, tuple is then similar to a row. Relvar is a specific relation type variable, and at every moment some relation of that type is assigned to it, even though the relation has none tuple.

This model was introduced to world by E.F.Codd, who worked in IBM's San Jose Research Laboratory in 1970's. Later other people as Chris Date and Hugh Darwen with their teams were the ones who developed and maintained relational model. Codd made his „12 rules“ (in fact 13, because they are numbered from zero to 12) to define what a database management system must fulfill in order to be accepted as relational. But in fact, none of used RDBMS nowadays comply to all 13 rules, the only example which does is Dataphor.

2.1.3 Structure

As was already said, when speaking about structure of a relational database, we use term table to visualize basic stone of database. It is mathematically incorrect term, in theory we use term relation. Now we will go through other basic terms of relational database, which will be shown on an example. Lets say we want to make a database of city citizens, where we want to know their name, surname, address and date of birth, further we assume, that each person has unique birth number. Each of these characteristics is called attribute. Each attribute can get one of different values (for address it would be one of city's streets). Complete set of these values that attribute can have is called a domain. Information that characterizes one of citizens, puts in relation different values of attributes from various domains. Such a group of attributes belonging together is called a n-tuple. Here is a complete mathematical definition of what has just been explained.

Given a collection of sets D1,D2,...,Dn (not necessarily distinct), R is a relation on these n sets if it is a set of orderes n-tuples <d1,d2,...,dn >such that d1 belongs to D1, d2 belongs to D2,..., dn belongs to Dn. Sets D1,D2,...,Dn are the domains of R, The value n is degree of R. [1]

In writen text we write simply R(A1, A2,...,An) if we want to describe a relation R with attributes A1...An. We can deduce few consequent rules from the definition, which are that because the relation's body is set on n-tuples, it is not ordered, from the same reason there are no duplicate n-tuples, and second, that some of attributes can be defined on the same domain, but names of attributes must be unique.

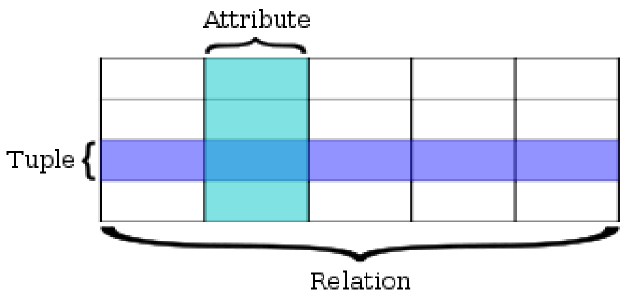

Figure 2.1: Relational database [14]

2.1.4 Constrains

Each database usually contains more than one table and information in these tables can relate to each other. For example we already have table of our citizens and now we also have table with city houses. Each house has different size, different status and different owner. And here we get to a relation between our two tables - house owner is always a citizen. So there is relationship „owns“ between those tables. In relational database this relation is purely logical, not physical (pointers) as before in pre-relational, and it is created upon equality of values in certain rows of both tables. In the example it will be owner and name. To make it possible to create such relations, we need to make sure that each row will have unique way how to identify it and then there must exist a link to this identifier in second table. In relational databases this is achieved by using keys. For identification it is a primary key and for links it is a foreign key. Data in every database is always somehow constrained. There are two types of constrains, general and specific. General constrains depend on usage of database and data is constrained in way of data type or minimum/maximum possible value. On the other hand, general constrains are the same for every database and they have effect on key attributes.

Candidate and primary key

Primary key is used to identify each row of a table, therefore it must be unique. We usually use logins, birth numbers or just integers as ids for primary keys. Sometimes there can be more candidates for primary key, so we have to chose on of these candidate keys to be primary. Attribute CK of relation R is called a candidate key if applies to these rules:

- Values of attribute CK in relation R are unique - there are no two n-tuples in relation with the value of this attribute.
- Attribute CK is specific and minimal - it is not composed of more other attributes and can not be divided into simpler

These rules must be valid in any time given. Primary key is then one of candidate keys, the rest of them are called alternative keys or secondary keys. We usually chose the simplest candidate key to become primary. Primary keys are used in other tables as links to its table, therefor not only that it has to be unique, but it also can not be unset or unknown, in terms of SQL we call such a value NULL, so primary key can never be NULL.

Foreign key

Attribute FK of relation R is called foreign key, if applies to these rules:

- Each value of FK is fully inserted or fully not inserted
- There is relation R2 with candidate key CK, that any value of FK equals to some of values from CK of n-tuple from that relation.

Again we can draw some conclusions. We see that foreign keys do not have to have a value, that might be case where for example some house is not owned by anybody, so the attribute owner will be empty. It is important that every foreign key really leads to some candidate key in other table, because that is the thing keeping database consistent and it is responsibility of administrator/programmer to secure it.

2.2 Tree databases

These databases, also knows as hierarchical, are databases, which were mainly used at the beginning of computer era, before times when the relational model became most common standard. As the name suggest, data in this model is stored in tree structures. From different point of view a tree allows us to store data in child-parent structure, which is referred also as 1-n structure, because each entry can have only one parent, whereas a parent can have multiple or no children. In this type of database, entries are connected by these parent-child relations, which are in fact simple pointers.

To look at hierarchical database in the same way as at relational one, table would be entity type, row a record and at last, column would be an attribute. These databases are not much used nowadays and if, then just for special models like geographic informational systems or file system data. Most used implementations of hierarchical databases in the present are Windows Registry developed by Microsoft and IBM's IMS.

2.2.1 XML databases

[11] XML database is another type of database model, this one is specific by storing data in XML format. This data can be queried as in other models, but can be also exported and serialized into other formats. We divide XML databases into three major groups:

- Native XML Database (NXD) - the basic unit of this database is an XML document; it does not require any underlying physical model as it can stand on a relational, OO database or any other kind of database or even just on indexed or compressed files; it defines XML document logical model, that at least must have elements, attributes, document order and PCDATA.
- XML Enabled Database (XEDB) - this kind of database has also a XML mapping layer, that manages the retrieval and storage of XML data. Mapped data is stored in a specific format, so the original XML meta-data can be lost. To manipulate this data we can use either special XML technologies as DOM, XPath etc. or SQL.
- Hybrid XML Database (HXD) - This kind of database can be treated as both NXD or XEDB, it is just on application which way it will prefer. Implementation of HXD is for example Ozone.

XML databases find their place with informational portals, product catalogues, business to business document exchange and many others. When applying XML databases for these solutions, they offer far bigger performance then relational databases and they are more convenient and easier to use, manage and expand.

XML

XML stands for Extensible Markup Language, which is nowadays format, that many applications use for encoding their documents. It was created to be both human-readable and machine-readable. Fundamental unit of XML document is an element, which is created using tags, a text construction that begins with „<“ and end with „>“, for example:

<person>bambusekd</person>

would be element representing a person bambusekd. Each element can have attributes, those can be either created inside beginning tag or as a separate couple of tags.

```
<person age="21" position="student">bambusekd </person>

or

<person>
    <name>bambusekd</name>
    <age>21</age>
    <position>student</position>
</person>
```

both examples provide same information.

XML was designed to be simple and to have wide scale of usability on the Internet. It is data format base on Unicode, so it can serve in any of world's languages. No matter the fact XML was designed for documents it found place also in many web services as representation of arbitrary data structures. XML is used in all major office tool applications as Microsoft office, OpenOffice, LibreOffice etc. XML has also been father to many other formats, that use XML syntax as XHTML, RSS and others.

2.3 Other databases

2.3.1 NoSQL

[9] Term NoSQL was first used in 1998 by Carlo Strozzi, who described his lightweight relational database as NoSQL because it did not follow design, structure and rules of typical SQL database. From then, you could not hear much about NoSQL, until a boom of social networks and a need for storaging huge amount of simple data came in 2009. In this year a guy from Last.fm made a statement, that NoSQL databases are those, that do not care about atomicity and consistency as traditional relational databases.

Main goal of NoSQL database is to provide lighter database, that would be faster and with higher availability, this is achieved by a model with looser consistency. That allows faster horizontal scaling. These databases consist of key-value entries, where relationships between them are not that necessary. NoSQL is focused mainly on adding data to database and retrieving them, not more. This with omission of relations allows these kind of databases to be used just for certain types of models. Such models can be millions of posts on social network like Facebook or Twitter, where there is no need of relations in between data, we just need to store them and retrieve them.

Term NoSQL does not mean that these databases are not SQL databases, because in fact some of them allow SQL-like queries above them, it rather means „not only SQL". Work on language, that would be specifically made to query NoSQL database has began in 2011. This language is called UnSQL (Unstructured Query Language) and it can query collections of documents, what in relational model would be tables and rows. Unfortunately UnSQL is not capable as SQL in manners of data definition, so there is no parallel query to SQL's CREATE TABLE/... in it.

Document store databases

Basic building stones of these databases, as name already says, is a document. As in relation model there are tables, here there are documents. Each implementation treats term document differently, but all of them encode data using one of formats as is JSON, XML, YAML or PDF. In comparison to relational model, records here do not have to follow strict scheme of data structure, that means that one type of entry can have on different occasions different sets of attributes. This allows to add any new information at any time without any problems with predefined data structure.

To address a document we use an unique identifier, that can be either name or URI. Usually these values are hashed to indexes, so data retrieval of any document if very fast. Basic query needed with this database is to retrieve document base on its identifier, however some implementations allow us to retrieve documents based on their content, but this depends on each implementation.

JSON - JavaScript Object Notation is a standard made to represent data structures and associative arrays. Although it is derived from JavaScript, which a scripting language used mainly in web environment, it is language-independent. JSON standard can be found in RFC 4627, it was first introduced in 2001 by Douglas Crockford and it is nowadays very popular serialization format used in server-client communication next to XML and others. It contains just few basis data types: number, string, boolean, array, object and null. The format was created in human readable form. Compared to XML it has lesser demands on data processing. Example of a person data encoded in JSON looks like this:

```
{
    "firstName":"David",
    "lastName":"Bambusek",
    "job": {
        "occupation":"student",
        "year":"3"
    },
    "age":"21"
}
```

YAML - is a recursive acronym for „YAML Ain't Markup Language", previously „Yet another Markup Language". It was brought on the light of the computer world in 2001 by

Clark Evans and it is another data serialization format, that is based on combined basis of programming languages as Python, Perl or C, XML and format of electronic mail. It is again representant of human-readable formats as was JSON or XML. YAML's main goal is to offer a way how to map high-level languages data types as are lists, scalars or associative arrays, to be easily modified, or viewed, so the ouput could be used for configuration files or document headers. Thanks to its syntax with whitespace delimeters it is very easy to work with YAML using grep or some of scripting languages as Perl. Same example used in JSON paragraph would look this way in YAML:

```
first_name: David
surname:    Bambusek
job:
    occupation: student
    year:       3
age:        21
```

Graph databases

[8] Graph databases have completely different structure than relational databases. They consist of nodes/vertexes, edges/relationships and properties/attributes. Graph database is index-free, this is due to fact that every entry has a direct pointer to his neighbouring element, so no lookups are needed. So as relational databases are based on mathematical theory of relations, graph databases are build upon a graph theory. Graph theory is useful in many ways, graph algorithms are used to find shortest paths, measure statistics like PageRank, closeness and it also offers a basis for high-performance databases.

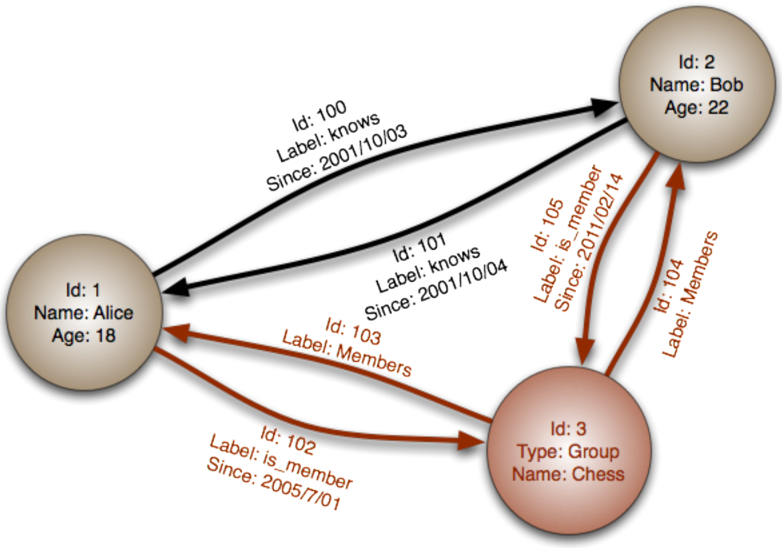

Figure 2.2: Graph database [13]

In graphs nodes represent entries, that means people, companies etc. Properties is some kind of information, that describes, details or specifies a node. If there is node David Bambusek, then one of properties could be „student“. Then we have last item of graph

- edge, which connects nodes with other nodes or nodes with properties, so in fact they represent the relationship between the two items they connect. These edges are the most important thing in graph model and carry the biggest amount of information. The way how to get some worthwhile information is to examine connections and properties of nodes we are interested in or of those we are led to on a way from starting nodes. In comparison to relational databases, graph databases are faster with smaller amounts of data as they do not need to perform join operations. It is better to use them in case, where it is known the database will change, grow and in some kind evolve, meaning we will add new kinds of nodes or add new information to existing ones. On the other hand, when we compare performance of relational and graph database on the same set of large data, relational wins.

2.3.2 Object oriented databases

[15] Object oriented databases differ from all other types of databases in their attitude of storing data. In other databases, data is stored using basic data types as integers and strings. In object oriented approach to databases data is stored in objects. Same thinking is used in OOP languages as C++ or Java. Objects were made so we could better reflect real world objects. Each object is compound of two basic parts. First is set of attributes, attribute is a characteristic of object, attribute can be either simple data type as integer or it can be again object. Second part is collection of methods, these define object behavior. All together object contains data and also executable code. Creation of new object is called instantiation. In this model, there exist so called classes, which are in fact templates for objects, they define their attributes and methods, but they do not contain data by themselves. To identify objects an OID in used in this model, which is unique ID of each object.

There are few fundamentals of object oriented approach, that are valid also in databases. First thing is that objects communicate in between themselves using messages, depending on received message object does something that is defined by its behavior. Next thing is encapsulation, that means that inner implementation of object is not visible from outside world, we can only see the interface. Therefore object itself or rather object data can be changed just by its own methods. This is so that there is no possible way how to make incorrect change to data stored in object. Also inheritance is very important, in fact it is one of the most useful features of OOP attitude. It is a way how to make relationships between objects and how to add some more specifications to new objects based on already created ones.

We should use object databases in cases when we have very complex data with a lot of many-to-many relationships. There is no sense in using them for small databases with simple relationships. Object database advantages over relational are that thanks to inheritance, we do not need that much code, the data model reflects the real world by using objects, navigation is much easier, there is reduced paging and finally it works very well with distributed architectures. On the other hand, relational model with tables is simpler, it is more efficient with simple data and there is generally bigger support, more software and more standards for relational databases, it is likely to say that object oriented approach is still bit wilder then more stable and standardized relational.

Chapter 3

Directory services

[6] [7] Directory services allow us to access data of directory type. By term directory we mean specialized database of telephone numbers, names, email addresses and others. These databases have their ancestors in printed directories of telephone numbers used by millions people around the globe. When computers came on scene during 1980's, also new network services using the Internet to create global telephone number directory showed up. These directories were base on ITU-T X.500 standards and later in 1990's, new standard was introduced, it was IETF Lightweight Directory Access Protocol (LDAP), which completely replaced X.500.

LDAP architecture, which is based upon X.500 architecture was created in 1997 and has had many actualizations. Today the most actual recommendations for LDAP are RFC 4510 and RFC 4511. Main goal for LDAP was to create a distributed directory service, that would treat all user equally and that would be easily extended. Basically for the same reason X.500 was created, but due too difficulty of implementation was never so successful as LDAP.

Architecture

X.500 defines directory as „a set of opened systems, cooperating in intention of preserving logical database of information containing set of object from the real world“ [6]. The architecture of database is hierarchical and is called Directory Information Base (DIB). It consists of entries, where each of them has set of information about itself, called attributes. Each attribute has a data type and a value. Each entry corresponds to some class of object and according to that class has different attributes, in other words object class defines structure of each entry.

We can imagine DIB as a tree, that is organized from top to bottom, where the most top entry represents the most generalized entry, for example country, going down the tree, we meet regions, then companies and so on. Each entry needs to be identified somehow, for this we use distinguished name (DN). It always consists of DN of its superior entry DN and its own DN.

Main difference between directory and relational database is in what we expect from them. Directories usually hold stable data, that are supposed to be read many times, rather then modified. We usually work with just one entry, so there is no need for operations like join or complicated search functions over more than one entry. Directory can contain duplicated data if it helps the performance of it. Most applications that use directories expect quick response from them.

Data stored in DIT can be physically stored on one server or can be distributed on more servers, it is wise to store them on more servers in case that we want to avoid overload of one server due to too many requests, we can also make a replication of some part of DIT, so request on this part can be divided onto more servers.

X.500 directory

This standard is ancestor and template for all later created directory services. It was created in 1988 and it is environmentally and architecturally independent. Its concept is quite similar to DNS, where it consists of entries representing all countries of the world. X.500 uses many protocols:

- Directory Access Protocol (DAP) - to access database of directory service
- Directory System Protocol (DSP) - to exchange information
- Directory Information Shadowing Protocol (DISP) - for data sharing between servers

But X.500 was too difficult to implement, therefore a lightweight version called LDAP was created. It is much faster, efective and uses just one protocol called also LDAP.

3.1 LDAP

[12] Lightweight Directory Acces Protocol (LDAP) represents not only internet protocol for directory access, but entire directory service. LDAP is based on ITU-T X.500 standard and is described by four so called models. Each of them describes one different view on the directory service:

- Name model - describes directory structure made of entries, that are identified by distinguished names (DN).
- Informational model - describes information which all together form so called directory scheme, those are data types, operations that can be done upon them and information how to store them. It also describes entries, attributes and their possible values.
- Functional model - describes operation done by LDAP protocol, where the most used operation is search and this model specifies where to search, using what keys and so on.
- Security model - describes how data is secured in database, that means if there is some authentication, encryption and access rights.

3.1.1 Name model

Name model describes organization of data and relationships between them. Name model describes how entries are stored into tree structure called Directory Information Tree (DIT). In this tree, each peak of the tree is entry, that can have ancestor/parent and descendant/child, to which they are connected by an edge. The structure of the tree corresponds to company structure, geographical structure or any other hierarchical structure, that usually reflects some real existing model. At the most top, there is a special root entry. To identify an entry we use distinguished name (DN), which is made from of all DNs of entry's parents. There is also special type of entry called alias, which refers to some other entry.

3.1.2 Informational model

Informational model defines entries, which as we already know are basic stones of DIT. Each entry contains information about itself, this information is stored using attributes, entry can have more attributes. Each attribute has a data type and a value, that can be either compounded value or just a simple value.

Lets make an example of Red Hat employee David Bambusek, entry will have an attribute cn, that goes for common name and mail, that logically stands for e-mail address, so in the directory there would be entry:

```
dn: cn=David Bambusek, dn=redhat, dn=com
cn: David Bambusek
mail: xbambu02 [at] stud.fit.vutbr.cz
```

Each entry belongs to a certain type of object class, these classes can be edited and created by database administrator. „Object class is a set of objects, that share the same characteristic.“[7]. Basic properties of class are that it defines which attributes entry derived from that class must and may have, defines which objects and entries belong to them, takes care about placement in DIT and checks operations being done over entries. As it is common in OOP attitude, classes can inherit some features of their parents. In LDAP we have 3 main types of classes:

- Abstract class- no entry can be based on this class, they only serve as templates for other classes.
- Structural class - each entry must be made based on at least one of these classes, every structural class is based (directly or indirectly) on the highest abstract class top.
- Auxiliary class - is used to extend attributes of entries.

Formal definition of class according to ABNF looks like this:

```
ObjectClassDescription = LPAREN WSP
    numericoid  ; object identifier
    [ SP "NAME" SP qdescrs ] ; short names (descriptors)
    [ SP "DESC" SP qdstring ] ; description
    [ SP "OBSOLETE" ] ; not active
    [ SP "SUP" SP oids ] ; superior object classes
    [ SP kind ] ; kind of class
    [ SP "MUST" SP oids ] ; attribute types
    [ SP "MAY" SP oids ] ; attribute types
    extensions WSP RPAREN
kind = "ABSTRACT" / "STRUCTURAL" / "AUXILIARY"
```

The only must parameter as we see is OID - which is identificator. Object class of our RedHat employee from example would look like this:

```
(2.5.6.6 NAME 'employee'
SUP top
STRUCTURAL
MUST ( cn \$ mail )
MAY ( departement \$ telephone ) )
```

That means there must be always employee's full name and mail given and we can optionally add department where he works and his telephone. We have two different types of attributes, they can be either user attributes, which are already presented cn, mail and others, these can be changed or modified, then there are operational attributes, that are generated automatically, are permanent and are used for administration, for example information of who created entry and when. Each attribute is identified by unique Object Identifier (OID). Basic types of classes and attributes can be found in RFC 4519.

3.1.3 Functional model

Functional model describes operations that can be done upon a directory, that means adding, modifying, deleting entries and querying them. Not only that it describes these operations, but it also defines way and scale of a query/search operation. If we have a big directory, it can be very useful to limit our query just on some part of directory. LDAP defines three types of search.

- base - search is done just in set base object
- one-level - search is done just in direct child of base object
- subtree - search is done in whole subtree of base object, with that object included

For a search operation we can use different filters and comparing rules. Not all data types support all kinds of comparing. For example operations like greater than/less than can be used only on attributes that can be ordered alphabetically/numerically/... Basic filtering rules are mentioned in a table 3.1.

Rule	Format
equality	(attr=value)
substring	(attr=[leading]*[any]*[trailing])
approximate	(attr =value)
greater than	(attr¿=value)
less than	(attr¡=value)
presence	(attr=*)
AND	(& (rule1)(rule2))
OR	(—(rule1)(rule2))
NOT	(!(rule1)(rule2))

Table 3.1: filtering rules for LDAP

There are few basic operations in LDAP that should be mentioned.

- Bind - is used to establish a connection between server and client, to agree on type of authentization and to login into directory
- Unbind - is used to end the connection
- Search - basic search operation, you must define the base object of search, the scope, the filtering rule, list of attributes we are interested in and maximal number of results we want to get
- Compare - is used to compare attribute values of set entries

- Modify - is used to change defined entry
- Add - is used to create new entry
- Delete - is used to delete an entry
- Abandon - cancels previous operation

3.1.4 Security model

Security model provides protection to data in directory against non authorized access. All LDAP servers have SASL autentization implemented. We can divide LDAP servers into three categories:

- public servers with read-only data with anonymous acces
- server supporting password autentization, MD5 SASL implementation is needed
- server supporting cryptation and authentization, they must implement TLS operations and authentization with public keys

3.1.5 LDIF

The LDAP Data Interchange Format (LDIF) is a standard, described in RFC 2849, that defines a plain text format for representing data stored in LDAP directories and LDAP operations like add, modify…From LDIF point of view, LDAP directory is a set of entries, where each of them has its own record.

Its birthplace was University of Michigan, where it was created by Tim Howes, Mark C. Smith and Gordon Good. LDIF has been extended and updated few times creating current standard specified in RFC 2849. We have already mentioned an example of LDAP entry, so here we have a modifying request on an entry, that adds e-mail address to an existing entry:

```
dn: cn=David Bambusek, dn=redhat, dn=com
changetype: modify
add: mail\newline
mail: xbambu02 [at] stud.fit.vutbr.cz
```

3.1.6 LDAP usage

As we said, directories are computer version of old telephone directories, so they most usually consist of information about people/organization/services, so the main usage is to get some contact information.

This is used for example with e-mail clients. When writing email, you insert name of recipient, mail client will ask LDAP server for his email address and that server will answer by exact address or in case there are more people with same name with all their addresses.

Quite similar situation is when using VoIP communication, each VoIP telephone has LDAP client, that can ask a LDAP server for a telephone number of a given person. Next example of LDAP usage is verification of users trying to access some services, most usually web service. Web client will ask user for login and password, send it to server, which will try to bind to LDAP server, which contains approved user logins and passwords, with this data and in case bind is successful, user is authorized. This approach is also used in Unix, where instead to authentize using /etc/passwd LDAP server is used.

3.1.7 LDAP implemantations

There are many implementations of LDAP servers, some of them are open source, some of them are commercial. Here is a list of the most known of them:

- 389 Directory server (Fedora Direcory Server) -it was developed by Red Hat. The name comes from the port number for LDAP, which is 389. This implementation is built in Fedora and is supported by many other distributions like Debian or Solaris.
- Active Directory - is a Microsoft implementation of a directory service, it was created in 1999 as part of Windows NT Server, it not only uses LDAP, but also Kerberos and DNS.
- Apache Directory - is an implementation of directory service entirely written in Java, it is an open source project created by Apache Software Foundation
- FreeIPA - is in fact combination of already existing projects, that provides managed Identity, Policy and Audit (IPA), it is focused on Unix computer networks. It uses Kerberos 5 for authentication Apache and Python for Web UI and 389 Directory server for LDAP. There is possibility to cooperate with Microsoft's Active Directory using Samba.

3.2 LDB

[10] LDB is a LDAP-like embedded database used in Samba project. Although it provides LDAP-like API, it is not LDAP standard compliant, its highest priority is to be compliant to Active Directory. LDB can be seen as a solution providing something between key-value pair database and an LDAP database. LDB is basically a module above TDB that manipulates key-value data into LDAP-like structure.

LDB is a transactional, that means it checks if any error occurred during changing the database data before committing it and if so, all the changes are backed, so the databasa is in the same state as before intended change. It is also modular, that allows any new functionality to be added or removed according to our needs on database performance. Available backends uses TDB, LDAP or SQLITE3.

LDB has many advantages of LDAP like custom indexes, it offers powerful search options, it is hierarchical and its structures can be easily modified or extended. On the other hand it keeps also some advantages of TDB as is fast search, all the data is stored in one file and it is easy to backup.

LDB enables fast searches thanks to function, that takes care of building indexes for it, when a new index is added, whole database is scanned so the indexes can be automatically rebuilt. Also there is no need for a schema, since any object can store arbitrary attribute-value pairs.

LDB has many powerful tools. In between them is worth mentioning ldbsearch and ldbmodify.

ldbsearch - its syntax is very similar to ldapsearch in LDAP, by using -H option, you define backend which should be used (tdb,sql, ldap,...) and then as in ldapsearch comes definition of the search scope, ehw base dn and a LDAP-like search expression.

ldbmodify - is a tool using known LDIF format, it allows you to explore and change a snapshot of the directory in a text editor, you can use filters to show just object you want to see, it can be also used to backup and restore database and it works against an LDAP server too.

3.2.1 TDB & DBM

As was already mentioned LDB is somewhere between TBD and LDAP. TDB is a successor of DBM database, made by Samba team. Its main difference from DBM is that it allows multiple writers to use database simultaneously and uses internal locking to avoid one entry to be rewritten by one user, while another one is working with it. DBM is a very simple database allowing to store any data in simple key-value structure. It was designed by Ken Thompson from AT&T in 1979. Each key is hashed to allow fast data retrieval. DBM uses fixed-sized buckets for primary keys, that split as database grows. Hash is usually directly connected to physical disk, so the retrieval can be very fast, because there is no need for any connecting or difficult querying.

Chapter 4

SSSD

[3] [2]

4.1 User login in Linux

In order to work with a Linux system, one must first log in. No matter if there is some GUI available or you log in using command line, you always have to enter your name and password. Term password covers not only ordinary text password, but it can be also fingerprint or any other device similar to it. Logging in has two phases. As first system needs to get information about user such as what his home folder is and as second, it is neccesary to authorize the user.

4.1.1 Identification

Information about users or for example hosts can be usually found in various files like `etc/passwd` or `etc/hosts`. Problem is that they are not on the same place, therefore some API is needed to work with all of them. In Linux, there is NSS(Name service switch) that serves this purpose. It is part of a standard C library, so it can be run on any system. It is a modular feature, where each module works with one source managing one type of object. For example one module will work with user from LDAP and second with passwords from `etc/passwd`. NSS is configurated using its config file in `etc/nsswitch.conf`.

4.1.2 Authentication

For authentication there is another API called PAM(Pluggable authentication module). It provides four main features - account, auth, seesion and password, where auth is the most important for us, because it tells us whether user can authenticate. PAM has also many modules, some of them are even based on anothers and complete programming is quite complicated. Most used modules are those providing connection to `etc/shadow`, LDAP, Kerberos, some additional modules offer advanced functionality as password quality checks.

4.1.3 Problems using NSS and PAM

Using NSS and PAM is possible for loggng into some system and this solution is working quite well, but there are few problems connected with it. For example when a situation of identity overlap occurs, when we have two domains with the same user, we have to somehow decide who is who and how he will be identified. Second big problem is how to query a remote server if a computer is currently offline. There are some options to solve this - a replica of LDAP tree can be made locally, information can be saved into `etc/passwd` or whole directory can be stored in cache, but all of these solutions are unhandy. Also with usage of NSS and PAM comes a lot of work with configurations for administrators, that are usually not very happy when given such a big amount of work.

4.2 Basic function

SSSD was created to solve all the problems mentioned above. Main idea of SSSD is to provide enhancements to Fedora or any other Linux distribution, that supports SSSD. First thing that SSSD offers is offline caching for network credentials. This is a big ease if you use a centrally managed laptops, because all the services as LDAP, NIS or FreeIPA will in fact work also offline. So what SSSD in fact does is that it provides access to SSSD cache for local services, cache stores information about identities from various providers as LDAP, Active Directory or Identity Management domain. SSSD used to be a client side part of freeIPA, but lately became separate project.

Next feature of SSSD is fact, that it reduces the overhead of opening new sockets for each query on LDAP, by using just one persistent connection to one or more LDAP/NIS servers, each acting as separate namespace. The only service that communicates with LDAP is SSSD Data Provider and that reduces the load on LDAP server to one connection per client. SSSD can be connected to various domains-sources of identities, where each of them can be connected with more servers, so when one of them is down, next server on list will be used.

Additional inovation, that SSSD brings is a service called InfoPipe, that works on D-BUS system. This service contains extended data information as preffered language or your profile image, which until now was mainly concern of various configuration files in user's home directory, which is not always available, due to mounting of home directory has not yet been done.

In summary, the benefits of SSSD are that laptop users can use their network logons even when they are offline, with SSSD you only need to manage one account. Just one service is needed to work with multiple identity and authentication providers. Developers will have access to InfoPipe, what brings new approach for extended user information, other services as FreeIPA, LDAP or NIS can take advantage of offline features thanks to the caching and the last, that it will provide FreeIPA client side software, for entering into FreeIPA domains. And the whole configuration of SSSD is matter of few lines compared to NSS and PAM together, so it is very easy to use for administrators.

4.3 Architecture

4.3.1 Processes

The SSSD is a set of four main processes, each of them has its own special function:

1. the monitor -is the process, which checks if other processes are running, it spawns them on the start and then re-spawns them if one of the periodical check shows that any service is not working.

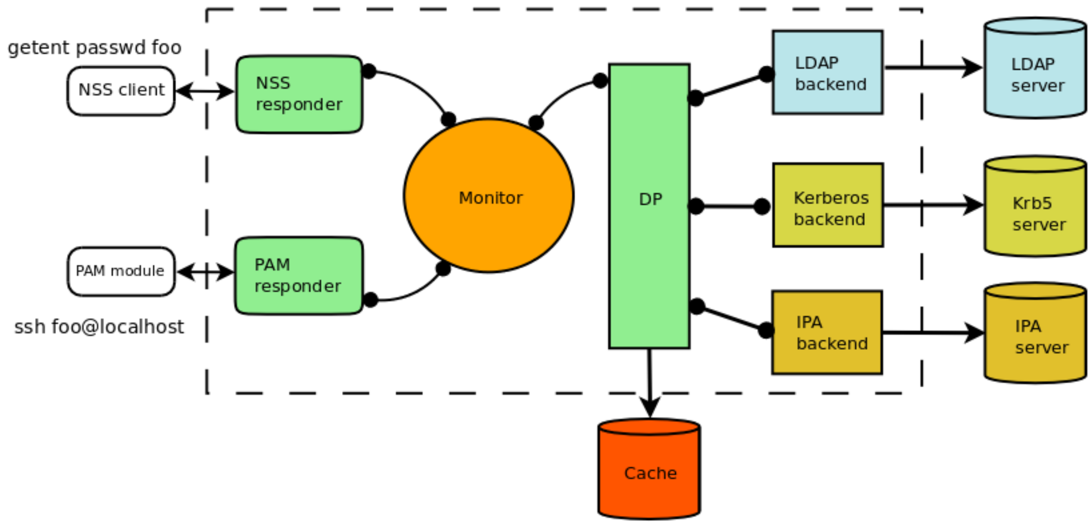

Figure 4.1: SSSD architecture [4]

2. a data provider - this process is responsible for communicating with various backends and populates cache with data obtained from them. For each remote server, there is one data provider process.
3. responders - are processes, that communicate with system libraries as NSS or PAM and try to give them data, that they are asking for, from cache. In case that data in cache is expired or is not there at all, it gives a signal to data provider to obtain it. When data is obtained, it is stored in cache, responder is informed about it, so it again goes into cache and gets the needed data.
4. helper process - there are some operations, that could be blocking, therefore SSSD performs them in special sub-processes, that are forked from the Data Provider.

4.3.2 Communication

D-Bus

For communication in between processes the D-Bus protocol is used in SSSD. Communication is done in form of sending messages. D-Bus can be divided into four primary components:

1. The D-Bus Server - is used for establishing connections. It can be identified by its address, that consists of a transport name, colon and an optional list of keys and values separated by commas.
2. The D-Bus Connection - these connections are peer-to-peer connections, so one end listens for method calls, and the other one of them initiates methods and vice versa.
3. The D-Bus Message - There are two types of messages:
    - one way messages - these are D-Bus Signals and D-Bus Errors, they usually carry a simple message from one end of connection to the other one. These are most often signals to stop/start service or to notify that some error has occurred.

    - answer messages - these are D-Bus Methods, their functions is to run a method on a remote process as it would be run locally, after calling the method, there can, but not necessarily has to, be an answer to it that goes back to end that called the method.

4. The D-Bus System Bus - is not used in SSSD, but it is part of D-Bus protocol. It was designed by the Freedesktop project and was created to handle multiple communication between system daemons.

S-Bus

To ensure that SSSD will have good performance, it works completely in non-blocking way with help of the tevent event loop library developed as part of the Samba project. To provide certain level of abstraction and in order to make possible to integrate D-Bus with the tevent, SSSD uses S-Bus - a wrapper created around D-Bus library.

Two processes in SSSD work as S-Bus servers, which is an abstraction of D-Bus server, they are identified by an UNIX socket, mentioned in the heading text, and they are:

- The monitor - it can call methods as „ping“ to check processes as was described above or „rotateLogs“ to rotate logs by force and others. Socket - /var/lib/sss/pipes/private/sbus-monitor
- The Data Provider - it calls different methods depending on the data type that is requested, for example the NSS can call method „getAccountInfo“. Socket - /var/lib/sss/pipes/private/sbus-dp_$ domain_name

Chapter 5

SSSD Database

SSSD database stores few different kinds of objects. There are 7 of them in total and in this chapter we will briefly introduce each of them. As was previously said, each object is an instance of a class that defines object's attributes. Attributes of objects like users and groups are very similar to those typically used in LDAP, so we can find a lot of similarity here. Complete list of attributes for these objects of LDAP character can be found in `ldap_opts.h`[1].

5.1 Users

User objects store basic information about users, so in fact a user object is a virtual identity of a real person. Every user can be member of multiple groups or netgroups.

`objectClass=user`

attribute	description
uid	user name
userPassword	user password
uidNumber	UID
gidNumber	GID
homeDirectory	home directory
loginShell	user shell
gecos	full name

Table 5.1: User attributes

[1] src/providers/ldap/ldap_opts.h

5.2 Groups

Users can be members of groups, groups can also belong into groups, so multiple nesting is possible. There is a tool `sss_groupshow` that can be used to display group members.

`objectClass=group`

attribute	description
cn	group name
userPassword	group password
gidNumber	GID number

Table 5.2: Group attributes

5.3 Netgroups

Netgroups are slightly different from groups, there are network-wide groups, that define set of users that have access to specific machines, set of machines with specific file system access and set of users with administrator privileges in specific domains. Netgroup is specified by a name and its members are in format of triples where one field is for machine, second for user and third for domain name.

`objectClass=netgroup`

attribute	description
cn	netgroup name
nsUniqueId	netgroup unique UID

Table 5.3: Netgroup attributes

5.4 Services

There is not much to say about definition of service, as the name speaks for itself, each has a name and runs on a different port.

`objectClass=service`

attribute	description
cn	service name
ipServicePort	service port
ipServiceProtocol	service protocol

Table 5.4: Service attributes

5.5 Autofs maps

Autofs map is a special feature used by automounter to automatically mount filesystems in response to access operations by user programs. When automounter is notified about attempts to access files or directories under selectively monitored subdirectory trees, it dynamically and transparently accesses local or remote devices.

`objectClass=automountMap`

attribute	description
cn	autofs entry key
automountInformation	autofs entry value

Table 5.5: Autofs map attributes

5.6 Sudo rules

Sudo rules define users who have been granted some kind of access, commands that are in scope of this rule and a hosts that this rules applies to. They can also contain some additional information, but it is mainly only „who can do what and where“.

`objectClass=sudoRule`

attribute	description
cn	sudo rule name
sudoCommand	command
sudoHost	host
sudoUser	user
sudoOption	option
sudoRunAsUser	run as certain user
sudoRunAsGroup	run as certain group

Table 5.6: Sudo rule attributes

5.7 SSH hosts

SSH is a cryptographic network protocol for secure data communication, that allows us to connect to remote machines. SSH hosts are then those machines we connect to. To connect to a host we need a key, which is exactly what we store in the database.

`objectClass=sshHost`

attribute	description
cn	ssh host name
sshPublicKey	public key
sshKnownHostExpire	time until entry expirates

Table 5.7: SSH host attributes

Chapter 6

Querying tool

6.1 Program specification

Querying tool is a console application called sss_query, its purpose is to query all kinds of data, that are stored in SSSD database and offer its users results of these queries. Application will be a part of tool package for administrating SSSD database. User can chose how wide his query will be, he can search just in one particular domain or in all of them that are available. It is possible to search for one exact entry, based on its unique identification, which varies depending on a type of an entry we are searching for or can provide information about all entries of the same type eg. users, groups etc. currently present in the database. For each entry type, there is a default set of attributes, that will be printed out, but if user wants to see just some of the attributes, he is free to notify the tool by adding them as an optional flag and application will provide just these to him.

Implementation language of the application is C.

6.2 Basic information and user interface

Application is able to answer queries on these types of objects:

- users
- groups
- netgroups
- services
- autofsm maps
- ssh hosts
- sudo rules
- domains
- subdomains

For each item there is a predefined set of attributes, that will be showed to user, in case he does not insert his own attributes as an optional parameter. Table 6.1 shows default attributes for each entry type. Application has two mandatory (6.2) flags and one optional (6.3) flag.

The application is a standard console application with no graphical user interface, all the communication between user and the tool is done upon the insertion of application's parameters and flags at the time of executing it. Result or multiple results if so, are displayed on standard output in plain text format, if there is no result or some problem occurred during run of the application an error message will be displayed. After completion of query, successful or not, an application is terminated, therefore if more queries are requested, we need to run the application once again for each query.

An example of communication between user and application looks like this:

```
Query on existing object
$ ./sss_query -u -N bambusekd -S uidNumber,name
$ > === Showing requested user ===
    UID Number(uidNumber): 00015487
    Name(name): David Bambusek

Query on non existing object
$./sss_query -u I name=filutam
$ > Object not found!

Application error
$./sss_query -s I gid=002587
$ > You can use GID just for identifying groups!
```

Entry type	SSSD internal macros	attributes
users	SYSDB_NAME, SYSDB_UIDNUM, SYSDB_GIDNUM, SYSDB_HOMEDIR, SYSDB_SHELL	name, uidNumber, guidNumber, homeDirectory, loginShell
groups	SYSDB_NAME, SYSDB_GIDNUM	name, gidNumber
netgroups	SYSDB_NAME, SYSDB_UUID	name, nsUniqueId
service	SYSDB_NAME, SYSDB_USN, SYSDB_SVC_PORT, SYSDB_SVC_PROTO	name, entryUSN, ipServicePort, ipServiceProtocol
autofsm maps	SYSDB_AUTOFS_ENTRY_KEY, SYSDB_AUTOFS_ENTRY_VALUE	name, automountIn-formation
ssh hosts	SYSDB_SSH_HOST_OC, SYSDB_SSH_KNOWN_HOSTS_EXPIRE, SYSDB_SSH_PUBKEY	sshHost, sshKnown-HostsExpire, sshPublicKey
sudo rules	SYSDB_SUDO_CACHE_AT_CN, SYSDB_SUDO_CACHE_AT_USER, SYSDB_SUDO_CACHE_AT_HOST, SYSDB_SUDO_CACHE_AT_COMMAND, SYSDB_SUDO_CACHE_AT_OPTION	cn, sudoUser, sudoHost, sudoCommand, sudoOption
domains	name, provider, SYSDB_VERSION	name, provider, version
subdomains	SYSDB_NAME, SYSDB_SUBDOMAIN_REALM, SYSDB_SUBDOMAIN_FLAT, SYSDB_SUBDOMAIN_ID	name, realName, flatName, domainID

Table 6.1: Default entry attributes

short opt.	long opt.	value format	description	example
-u -g -n -s -m -t -r -d -a	–user –group –netgroup –service –autofsmap –sshhost –sudorule –domain –subdomain	no value	type of object to be searched	-u
-N -U -G -P -I -A	–name –uid –gid –port –ident –all	[name] [uid] [gid] [port] [identificator]=[value] no value	entry identification query all objects of chosen type	-N bambusekd –uid=32568

Table 6.2: Mandatory flags

short opt.	long opt.	value format	description	example
-S	–show	[attribute_name]+	If set, just specified attributes will be displayed, otherwise attributes from default set 6.1 will be displayed.	-s name, mail

Table 6.3: Optional flags

6.3 Application architecture

6.3.1 Inicialization

The program starts by typical argument processing with help of library `popt.h`[1], that offeres very user friendly set of functions and preset macros to easily process arguments given by user through command line. Next phase is to validate given arguments, because logically there are many forbidden combinations, because each of object can be identified just by few specified identifiers 6.4. To process more complex arguments as are attributes that user wants to see on output so as to process type of identifier which will be used, serves a function `parse_attributes`, that simply parses all the arguments and depending on input determines what kind of identification will be used and saves this information into main system structure `query_tool_ctx`, which is used not only to store this information, but mainly core data as is a link to confdb database, information about domains and list of system databases, all mentioned later.

Object	Identifiers
user	name, UID
group	name, GID
netgroup	name
autofsm	name
sudo rule	name
service	name, port
ssh host	name

Table 6.4: Object identifiers

For a memory allocation we do not use standard C `malloc()` and his relative functions, but a library `talloc.h`[2], which is a hierarchical, reference counted memory pool system with destructors, that brings easiness for otherwise complicated way how to allocate and free memory in C, which in bigger project grows to very messy thing and can be a source of many memory leaks. Thanks to hierarchical system, we only need to take care of deallocating the root object, that is connected to its children by links created upon their allocation and the library takes care of deallocating the whole connected tree of allocated memory.

As next we procede to connection to SSSD database, from here reffered to as „sysdb“, by using function `init_domains()` borrowed from another tool `sss_cache`. This function firstly establishes connection to confdb, which includes all neccessary information and settings to enable connection to sysdb and as a next step finally connects to sysdb. All this functionality is done by functions already implemented in SSSD[3].

Attributes saved in SSSD database are identified by names(strings) written rather in computer-like style, so for example `UID number` inside SSSD is identified as attribute `uidNumber`. But these names would not be very nice for users to read on output, that is reason why all attributes that sss_query offers to be displayed are mapped to human readable form. For this approach we use a hash table, which key-value entries have format of:
`<string, string>(SYSDB_<attribute>, human readable SYSDB_<attribute>)`
Thanks to use of hash table, we can later profit on very fast and simple way how get the

[1] http://docs.fedoraproject.org/en-US/Fedora_Draft_Documentation/0.1/html/RPM_Guide/ch15s02s02.html
[2] http://www.talloc.samba.org
[3] sssd/src/db/sysdb.h

right translation for SYSDB attribute to human readable form. To use and work with hash tables, we use library called ding-libs(libdhash).

Now we are in stage where we already know what type of object we will be looking for and we checked whether user also used correct identifier, so now the only thing that is left is the search for object itself.

6.3.2 Query on basic objects

In this stage we get to querying itself. There are many functions already implemented in SSSD that offer a way how to get information about objects stored in database. If we want to make a query, we need to call some of lbd functions on the lowest level, since ldb is the framework that allows us to work with database. The building stone of every functions that gets data from sysdb is `ldb_search` that is very similar to `ldap_search`, so we need to pass base dn, search scope and filters (3.1) as arguments to it and it will provide us with results of search as its return value in special structure `ldb_result`. This structure contains a pointer to array of the results from the search and is easily accessible for further work.

There are many functions that are wrappers around this basic and very general function for most of object in sysdb, each of these functions has its own specialties as each object has different unique identifier and needs to be treated differently in means of types of attributes it has or where in sysdb is this object located. These functions are widely used is sss_query and in cases where these functions for certain object are not available, sss_query uses basic `ldb_search` to get results.

The only difference between functions working with one certain object and those working with whole class of objects is, that there have not been implemented any functions, to get information about group of objects in SSSD, therefore `ldb_search` is always used in this case. We handle the results in basically the same way, we just have to iterate through complete list of results.

6.3.3 Domain and subdomain information

Different approach is used with domains. Since we have all the data about domain or domains stored in special structure since initialization, we do not need to make any additional queries on sysdb.

It is quite easy with subdomains again, each structure with domain information includes also pointer to array of subdomains, that are in every view similar to domains, just their location in tree hierarchy is one level lower than domains. It is worth mentioning, that subdomains can again have subdomains, so there can be whole nested structure of subdomains.

6.3.4 Printing results

Now when we already have our query results, we just need to display them to users. Because sss_query works as an console application and therefor lacks GUI, results are printed in text form on standard output. If user did not specified attributes he would like to see on output a default set of attributes will be printed out. In case of query on all objects of certain type, results are printed out one by one, visibly divided and provided with label saying from which domain they come from and what is their number in list of results.

Print function is quite straight forward, we have an array of attributes that will be displayed on the output. We iterate through them and with each we first find the human readable name, that will go on the output together with its sysdb internal name and of course its value . This value is gathered from the result message that `ldb_search` gave us using another ldb function `ldb_msg_find_attr_as_string` and then the complete information about attribute is printed out.

Nevertheless the output is done just in a text form, sss_query tries to format and display results or any massages in a way that it looks nice and it is easy to take in. So little bit of „ascii graphic“[4] is used to separate results and to create a header for each one, so the user is provided with all the necessary information in understandable and easy to read form.

6.4 Tests

Find user - all attributes

```
$# ./sss_query -u -N bambusekd
$# === Showing requested object ===
Name(name):   bambusekd
UID number(uidNumber):   1063200001
GID number(gidNumber):   1063200001
Home directory(homeDirectory):   /home/bambusekd
Shell(loginShell):   /bin/sh
```

Find user - using FQ name

```
$# ./sss_query -u -N bambusekd@example.com
$# === Showing requested object ===
Name(name):   bambusekd
UID number(uidNumber):   1063200001
GID number(gidNumber):   1063200001
Home directory(homeDirectory):   /home/bambusekd
Shell(loginShell):   /bin/sh
```

Find user - just UID and name

```
$# ./sss_query -u -N bambusekd -S uidNumber,name
$# === Showing requested object ===
UID number(uidNumber):   1063200001
Name(name):   bambusekd
```

Find all users

```
$# ./sss_query -u -A
$# === Showing all users in domain example.com ===
```

[4]Graphical object made just from ascii characters

```
=== entry: 0 === domain: example.com ===
Name(name):   bambusekd
UID number(uidNumber):   1063200001
GID number(gidNumber):   1063200001
Home directory(homeDirectory):   /home/bambusekd
Shell(loginShell):   /bin/sh

=== entry: 1 === domain: example.com ===
Name(name):   krausj
UID number(uidNumber):   1063200004
GID number(gidNumber):   1063200004
Home directory(homeDirectory):   /home/krausj
Shell(loginShell):   /bin/sh
```

Find all users - just UID and shell

```
$# ./sss_query -u -A -S uidNumber,loginShell
$# === Showing all users in domain example.com ===

=== entry: 0 === domain: example.com ===
UID number(uidNumber):   1063200001
Shell(loginShell):   /bin/sh

=== entry: 1 === domain: example.com ===
UID number(uidNumber):   1063200004
Shell(loginShell):   /bin/sh
```

Show domain info

```
$#./sss_query -d -N example.com
$# Domain name: example.com
Domain provider: ipa
Domain version: 0.14
```

Error - no object selected

```
$# ./sss_query
$# Please chose one type of object
```

Error - too many arguments

```
$#./sss_query -u -g
$# Please chose one type of object
```

Find non existing object

```
$#./sss_query -u -N smithj
$# === Object not found! ===
```

Find object in non existing domain

```
$#./sss_query -u -N bambusekd@fit.cz
$# === There is no domain fit.cz ===
```

Chapter 7

Conclusion

Reader of this thesis was introduced with overall information about databases, their different models with focus on those that are used the most these days, but we also did not forget to mention those types that are not that widely used, but are slowly finding their place in the modern, and in every moment developing, world of computers. We stopped for more detailed information about LDAP and LDB and moved to description of SSSD, its architecture, functions and its database.

The main output of this thesis is a tool for querying SSSD database, which is able to query all different types of data stored in SSSD database and to offer SSSD administrators the important information about them. This tool was developed with emphasis on very easy usage and therefore offers very simple UI which allows very complex control of application. I strongly believe that this tool will ease work of many people dealing with SSSD and provide all the functionality that was expected.

This application offers a lot of functionality, but there is a space to merge it with other already existing tools for SSSD administration so as with other tools, that has not yet been developed and might be needed. Therefore its current stage might not be ultimate application will change in future as will SSSD with newer versions that will be introduced.

Bibliography

[1] C.J. Date. *An Introduction to Database Systems.* Addison-Wesley Publishing Company, 1979. ISBN 0-201-01530-7.

[2] Red Hat. *Fedora documentation.* Red Hat.

[3] Jakub Hrozek. *SSSD Wiki - Design documents.* Red Hat.

[4] Jakub Hrozek and Martin Nagy. Freeipa and sssd. Presentation at Red Hat Developers' Conference, 2009.

[5] Richard D. Irwin. *Database Management - Theory and Application.* Irwin, 1990. ISBN 0-256-07829-7.

[6] ITU-T. The directory - overview of concepts, models and services. *X.500 (2005)*, August 2005.

[7] ITU-T. The directory: Models. *X.501 (2005)*, August 2005.

[8] Peter Neubauer. Graph databse, nosql and neo4j, May 2010.

[9] Pramod Sadalage and Martin Fowler. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence.* Addison-Wesley, 2012. ISBN 0-321-82662-0.

[10] Simo Sorce. Ldb and the ldap server in samba4. Samba experience, 2006.

[11] Alex Tatiyants. Xml databases, 2012.

[12] M. Wahl, T. Howes, and S. Kille. Lightweight directory access protocol, 12 1997. RFC 2251.

[13] Wikipedia. Graph database, 2013. [Online; accessed 22-April-20013].

[14] Wikipedia. Relational database, 2013. [Online; accessed 22-April-20013].

[15] Kim Wong. *Introduction to object-oriented databases.* MIT Press, 1948. ISBN 0262111241.