Intelligent System Supervision

Course details	
Course ID	VIMIA370
Instructor	Zoltán Micskei (http://mit.bme.hu/~micskeiz)
Course website	http://mit.bme.hu/~micskeiz/education/irf/eng
Lessons	Consultation lesson on every odd Thursday 10:15-12:00, IE412 room
Description	The course introduces the basics of IT system management, and the connection between software development and IT management. The course details typical management tasks and related technologies (e.g., configuration management, monitoring).
Requirements	Signature: - completion of two home assignments - every home assignment can get 0-5 points - at least 2 points shall be obtained from each assignment Exam period: - oral exam
Grading	Final grade (average of two home assignments) * 0,4 + 0,6 * oral exam

1.1 Topics

Throughout the semester the course will introduce the following topics:

Topic	Details
Modeling IT systems	Modeling basics, data models, metamodels, UML static modeling
Scripting languages	Scripting languages, Linux command line, the Python language
Directories	Storing user information, LDAP protocol and LDAP directories
Configuration management	Modeling configuration data, the CIM specification, accessing CIM data through the protocols of WBEM, CIM-XML, WS-Management
Virtualization and cloud computing	Virtualization basics, cloud computing definitions, types of cloud services

1.2 Home assignments

During the term weeks two home assignments shall be solved. Each home assignment consists of writing a script accomplishing an IT management task.

- Home assignment 1: on the topic of directories, published on the 6th week.
- Home assignment 2: on the topic of configuration management, published on the 9th week.

1.3 Exam

In the exam period an oral exam shall be taken. The topic of the exam consists of the topics of the course and the materials given in the handouts of the lectures.

2 Modeling IT systems

This topic will be about modeling parts of an IT infrastructure and creating data models in UML.

2.1 Reading material

[1] Refresh what you have learnt in the "Software Technology" course (VIIIA217), especially the part concerning UML.

[2] Kirill Fakhroutdinov. UML Diagrams. website, URL: http://www.uml-diagrams.org/

This is a good reference website about UML. For the current course, we will only use the basics of Class diagrams to create so called domain models.

[3] IEEE. "Guide to the Software Engineering Body of Knowledge". Ch. 10 Software Engineering Models and Methods, Sec. 1-3, URL: http://www.computer.org/portal/web/swebok/home

This is an overview about modeling in software engineering.

Additional reading (optional):

[4] J. Ludewig. „Models in software engineering – an introduction". Software and Systems Modeling 2(1), 2003, pp. 5–14. DOI: 10.1007/s10270-003-0020-3

This is a research paper about modeling in general and the different kinds of models used in software engineering.

2.2 Concepts and technologies

After reading the materials and preparing on the current topic, the students should understand the following concepts and technologies:

- what is a model, metamodel, abstraction and concretization, defining modeling languages;
- UML Class diagrams (especially multiplicity, visibility, association, generalization).

2.3 Exercises

The goal of this topic is to be able to model simple IT systems, and to be able to read more complex models, which will be introduced later in the course.

2.3.1 Simple computer systems

Exercise: the following picture depicts a system with several elements and their properties. Create a UML class model that can be used to represent such systems.

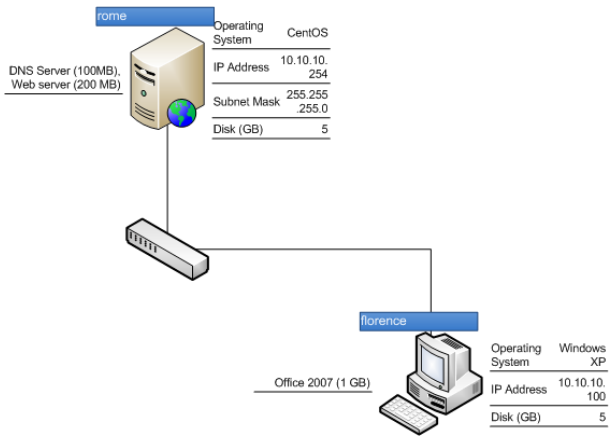
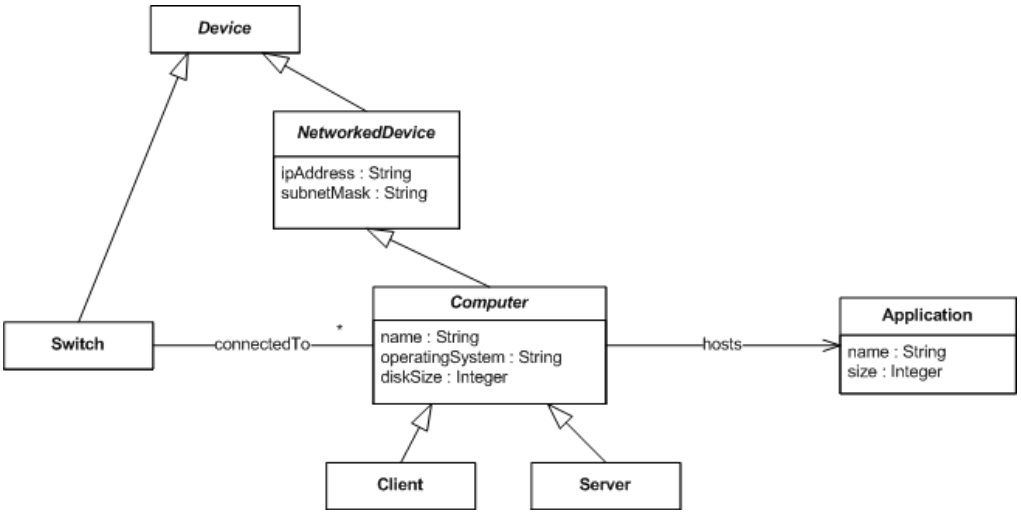


Figure 1: Simple computer system

Several, correct models can be created, but pay attention that the model should be able to express every detail depicted on the above picture.

Solution: One possible solution can be the following (other models can be also possible depending on modeling style, the purpose on the model etc.).

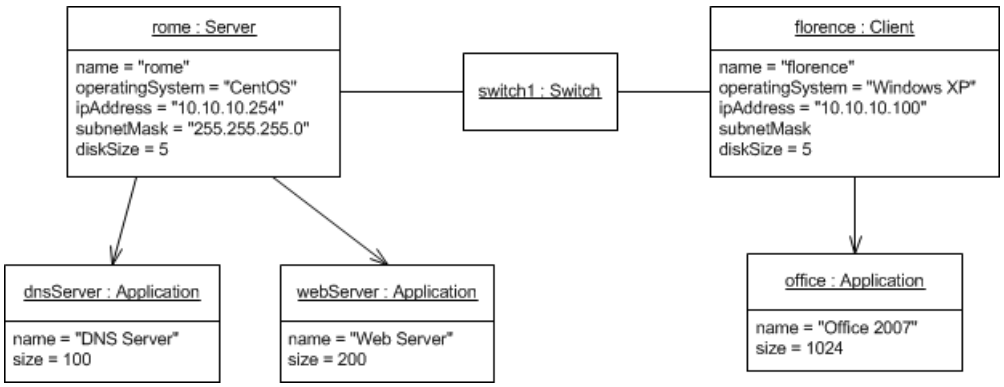
Class level ("metamodel"):



Note the followings:

- Client and server are differentiated because different drawings are used for them.
- Abstract classes are introduced (Computer, NetworkedDevice) to collect common properties. In modeling tools abstract classes are denoted by italic name, on paper this can be expressed using the «abstract» stereotype.
- Simple data types are used (Integer, String), later they can be refined to more complex types.

Instance level ("model"):



This model captures all the relevant data from the figure.

- Even on this simple model checks can be performed, e.g., the subnet mask for the machine named florence is missing.
- When creating instance models pay attention to name every instance uniquely.

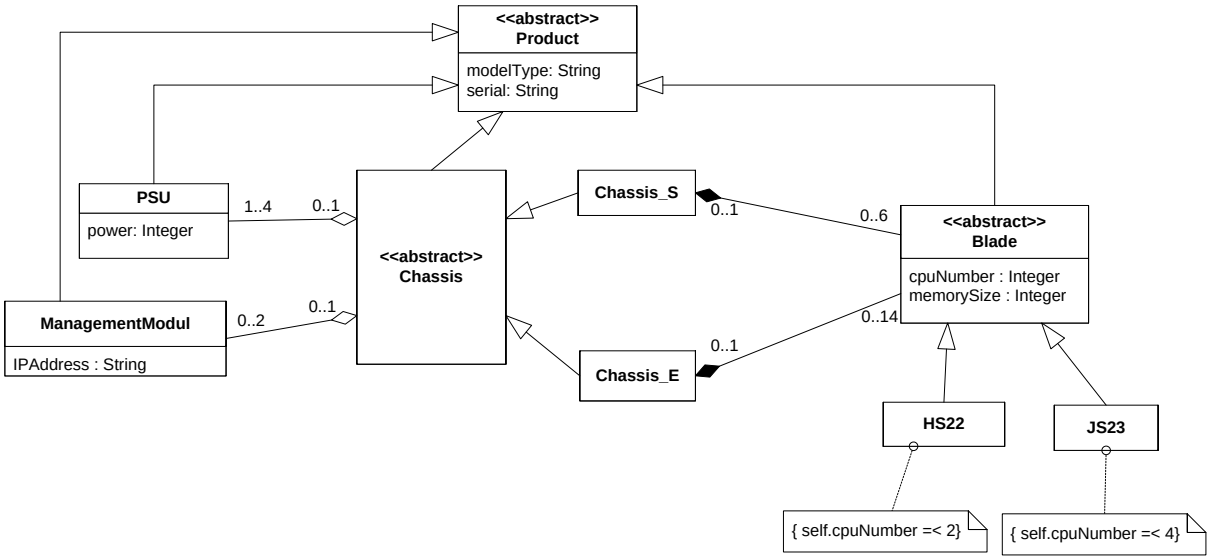
2.3.2 Complex example: BladeCenter

a) Create a model that can express information about BladeCenter systems (a special, dense computer server system). A BladeCenter system consist of a chassis, a chassis can contain blade servers. Currently we are dealing with chassis types E and S, type E can contain 14, while type S can contain only 6 blades. The chassis and blade models are identified by their model numbers, and the individual products are uniquely identified by their serial numbers. The chassis could include the following other elements: power supply units (at most 4 numbers, there are different models with varying power) and at most two management units. The system can be remotely managed through a management unit, a unit can be accessed through its IP address. We would like to store the following information regarding a blade: number of CPU and size of its memory. Currently we have two blade models: JS23 with 4 CPU sockets and HS22 with 2 CPU sockets.

b) After creating the above model, create an instance model (an object diagram) that represents the following system. We have a type E chassis which model number is 8677-3TG. The chassis contains two 2000 Watt power supply unit (model number 74P4452) and one management module, which has not been initialized yet. There is only one blade in the chassis, a number 7996-60 blade with type JS23 containing 2 CPUs and 64 GBs of RAM. After creating the instance model, identify those details, which were not given in the text but can be represented in the model.

Solution: A possible solution can be the following.

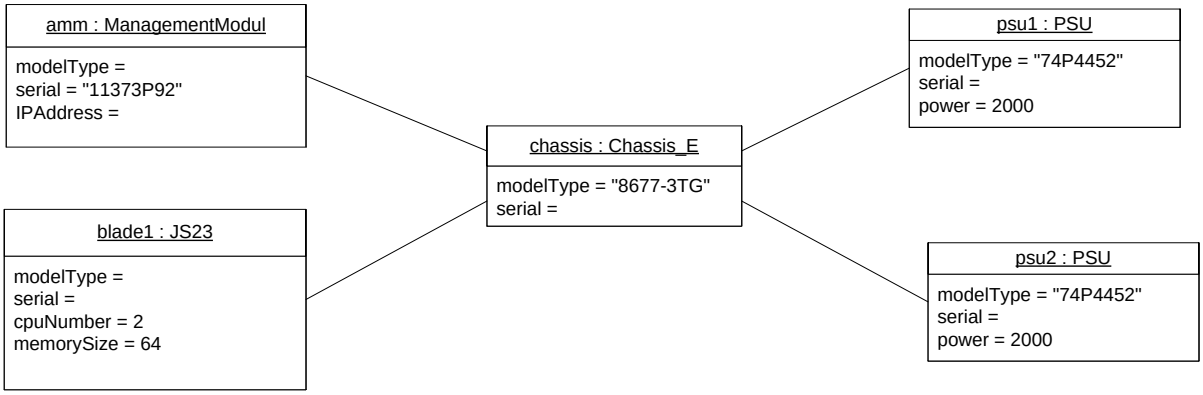
a)



Notes:

- We created an abstract class Product, because every other element should have serial number and model type.
- Pay attention to using a consistent naming convention (e.g., using PascalCasing)!
- In the current abstraction level we create conceptual or data models, this there is no need to include visibility for the attributes.
- If possible, add multiplicity to the association ends as it introduces further constraints.

b)



The important point in creating an instance model is that it should conform to the model defined in the previous exercise.

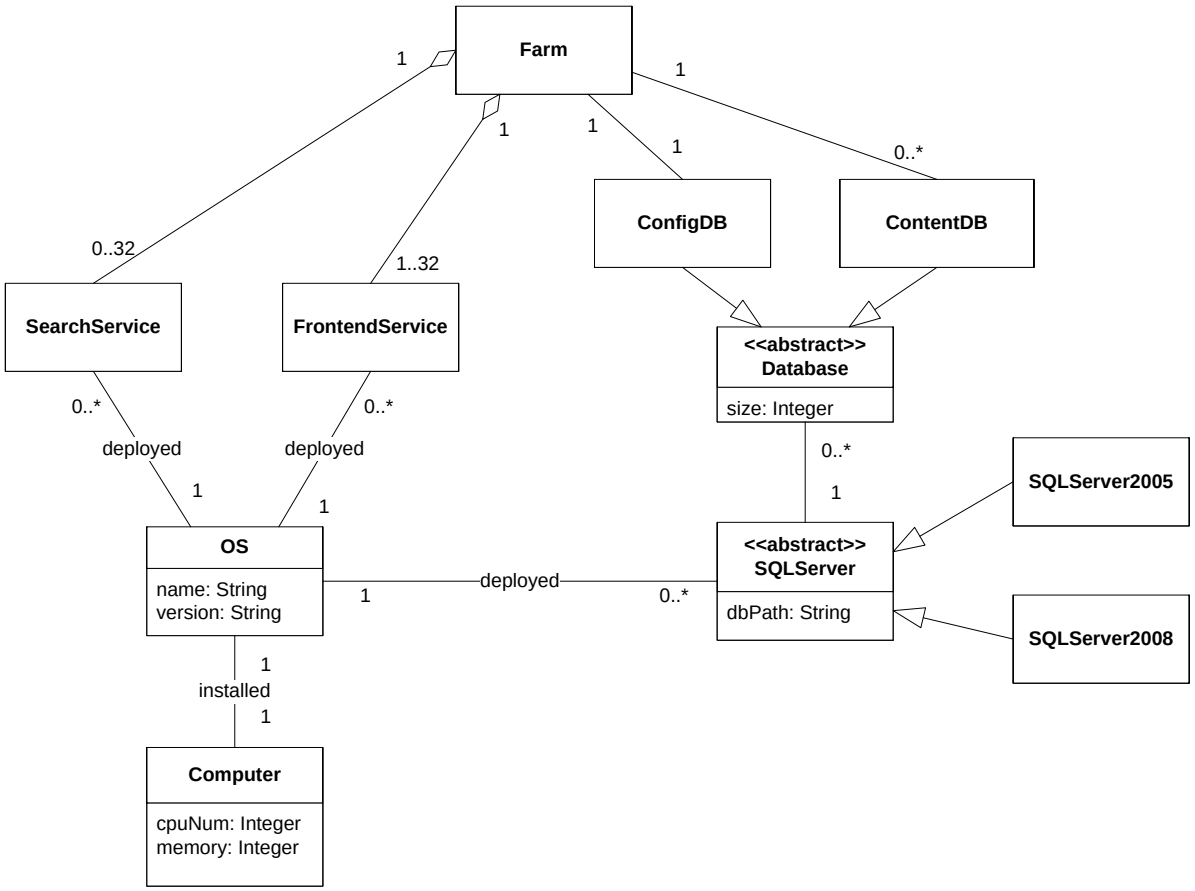
2.3.3 Complex example: SharePoint

a) We are developing applications for the Microsoft SharePoint platform (in a nutshell it is a product to create enterprise portals). We would like to create a metamodel that can be used to model the infrastructure used in the development and test teams. SharePoint offers flexible deployment options. The basic unit of the deployment is called a farm. A farm contains at least one web frontend service and optionally one or more search services. The frontend and search services can be installed to the same computer. At most 32 frontend service instances and at most 32 search service instance can be in a given farm. We would like to include in the model the followings: on which computer is a service installed, what operating system is installed on a computer, how many processors and how much memory is in each computer. Moreover, a farm needs databases to store its data. Exactly one configuration database and zero or more content databases are needed. We would like to store the size of each database. Databases can be stored in a SQL Server 2005 or a SQL Server 2008. Pay attention to include the multiplicities of the associations in the model!

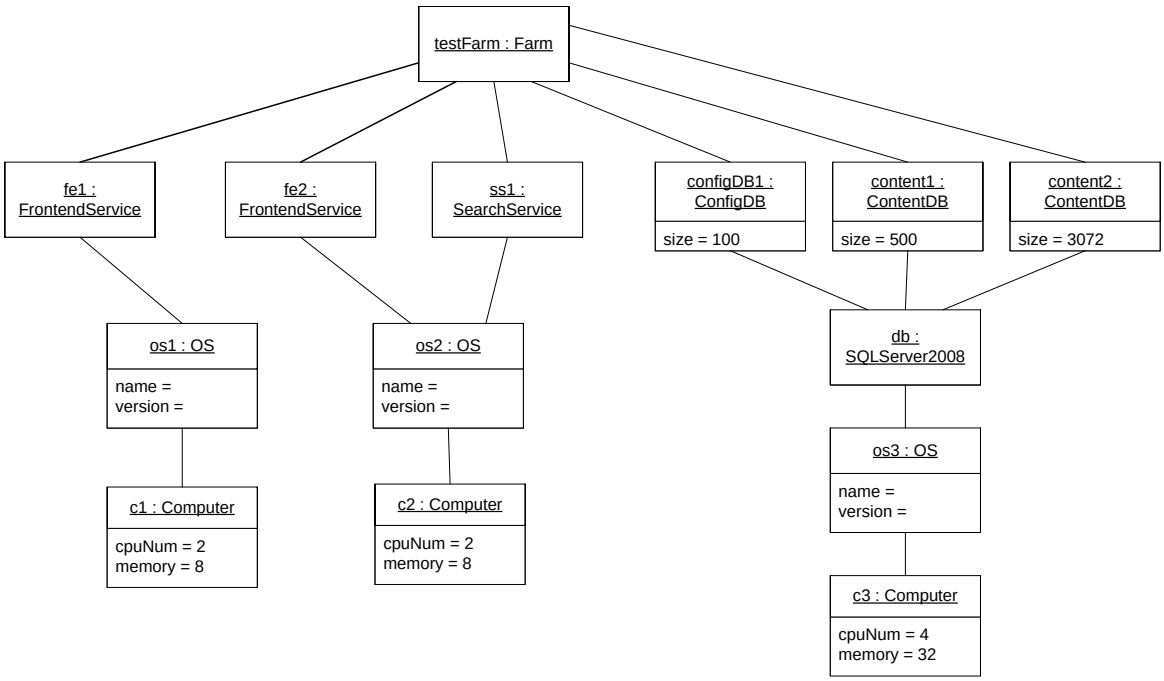
b) Create an instance of the above metamodel to represent a medium sized test infrastructure. There are two frontend servers in the farm, one of them is also hosting a search service. There is also a database server with SQL Server 2008 installed hosting a 100 MB configuration database and a 500 MB and 3 GB content database. The database server has 4 processors and 32 GB RAM, while the two frontend servers have two processors and 8 GB of memory.

Solution: A possible solution can be the following.

a)



Of course, there are other possible solutions, e.g. there can be an abstract Service class, different service types can be represented with distinct associations going between the farm and an OS or a computer, etc.



2.3.4 Complex example: Virtualization

a) We would like to develop a system that can manage hypervisors and virtual machines in a cross-platform way. (A hypervisor is a kind of a small operating system that creates and runs virtual machines on a physical computer.) Our first task is to create a UML model that calls the important concepts of the domain.

   There are hypervisors, which have names and version numbers. We currently support two hypervisors (VMware ESXi and Xen). There are virtual machines, which have operating systems installed. For the operating systems currently its manufacturer and version is important. The model should include the information on which hypervisor which virtual machine is deployed currently. Hypervisors are installed on physical machines. For the physical and virtual machines, we would like to store their number of processors and the size of their memory. Moreover, virtual machines may have virtual disk, which have size expressed in GB.

b) Create an instance model for the above model that contains at least two hypervisors and three virtual machines.

3 Scripting languages

This topic will introduce scripting languages using the Python language as an example. Scripting languages could be used in many situations, they offer a quick solution for automating repeating tasks, they can be used on the fly for complex problems (renaming a large number of documents, downloading several files, extracting information from text files, etc.), or they be used to create components in applications (e.g., scripting languages are used frequently to develop administration interfaces).

Some specialties of scripting languages compared to compiled programming languages (C, Java):

- usually an interpreter is processing the script;
- the code can be processed line by line;
- in many cases variables are dynamically typed.

This makes scripting languages easy to learn and create programs quickly.

Notable examples of script languages used nowadays include[1]:

- Bash (the default shell in several Linux versions);
- JavaScript, Ruby, PHP (languages used frequently in web applications);
- PowerShell (the new scripting and automation framework in Windows).

During the course, we will use the **Python** scripting language, because

- it resembles languages previously covered in the curricula;
- its documentation is very detailed and its community is very active;
- it is widespread, numerous open source projects and companies use it (e.g. it is one of the languages Google uses internally).

3.1 Reading material

[5] Python Software Foundation. "The Python Tutorial". http://docs.python.org/2/tutorial/

This is the official tutorial. You can come back here later for explanation of various topics.

[6] Google. "Google's Python Class". 2012. URL: https://developers.google.com/edu/python/

Google has put together an excellent class on introducing Python with learning videos and exercises. Start with this course; it will walk you through all the basics. You can watch also videos explaining the different concepts.

[7] Guido van Rossum, Barry Warsaw. "Style Guide for Python Code", 2013. Python Enhancement Proposal PEP-8, URL: http://www.python.org/dev/peps/pep-0008/

This is the official style guide, which describes how you should format your Python code to make it readable and easy to understand. After you solved a few basic exercises, read this document and try to stick to the conventions outlined in it, as it will help you to develop better code.

3.2 Exercises

As with any other programming languages, the best way to learn the language is to experiment with it, try to write code or solve simple problems.

[1] See: http://en.wikipedia.org/wiki/List_of_programming_languages_by_category#Scripting_languages

In the course only the basics will be required (variables, lists, control flows, calling external commands, creating simple scripts), we will not cover creating new classes or modules.

1. Get a working Python environment.
2. Solve the problems in the Python Exercises part of the Google class. The downloadable material contains the solutions also, thus you can check later your code.
3. Create a script named `collect.py` that does the following. It gets the path to a directory as its only input. After running, the script should write out to the standard output the number of subdirectories in the input directory (counting recursively), and the file extension, which was the most common in the files contained in the directories. The script should check whether the input directory exists, if not, it should produce an error. If there are several file extensions with maximal occurrences, the script should write out all those extensions.

3.3 Example

This is a short example that demonstrates how Python code looks like.

```
import argparse
import sys
import os.path

# Argument parser initialization
parser = argparse.ArgumentParser();
parser.add_argument("name", help="The name to be greeted.", type=str)
parser.add_argument("-q", "--quantity", help="Amount of greetings.", type=int, default=1)
parser.add_argument("-f", "--file", help="The output is written to this file.", type=str)
parser.add_argument("-X", help="Force to overwrite output file", action="store_true")
args = parser.parse_args();

# Argument checking
if args.quantity < 0:
    print("ERROR: Quantity shall be a positive number.")
    sys.exit(1)

# Print greeting to screen
i = 0
while i < args.quantity:
    print("Hello ", args.name, "!", sep="")
    i = i+1

# File handling
if args.file is not None:
    if os.path.exists(args.file) and args.X == False:
        print("ERROR: Output file already exists. Use -X to overwrite.")
        sys.exit(2)
    else: # Write to file
        f = open(args.file, 'w')
        i = 0
        while i < args.quantity:
            f.write("Hello " + args.name + "!\n")
            i = i+1
        f.close()
```

4 Directories

The next topic will introduce directory services, and more specifically LDAP-based directories. A directory is a special database for storing information about users, groups or computers. Usually the directory is a key part of any IT infrastructure, as it is the basis of any central authentication or authorization method. The most widely used specification for the structure of such databases and the protocol to access the directory is the *Lightweight Directory Access Protocol (LDAP)*.

4.1 Reading material

[8] Zytrax.com. „LDAP for Rocket Scientists", Open Source Guide, version 0.1.14, URL: http://www.zytrax.com/books/ldap/

It is one of the best online guides for LDAP. Read Section 1 "Overview & Concepts" (except 2.5, 3.5 and 3.6).

Additional reading (optional):

[9] Internet Engineering Task Force. „Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map", RFC 4510, June 2006. URL: https://tools.ietf.org/html/rfc4510

This is a pointer to the official LDAP specifications. At least have a look once at an RFC, to get an impression how the key Internet technologies are defined.

4.2 Example of an LDAP directory

The following picture presents a simplified LDAP directory to help understand the key concepts.

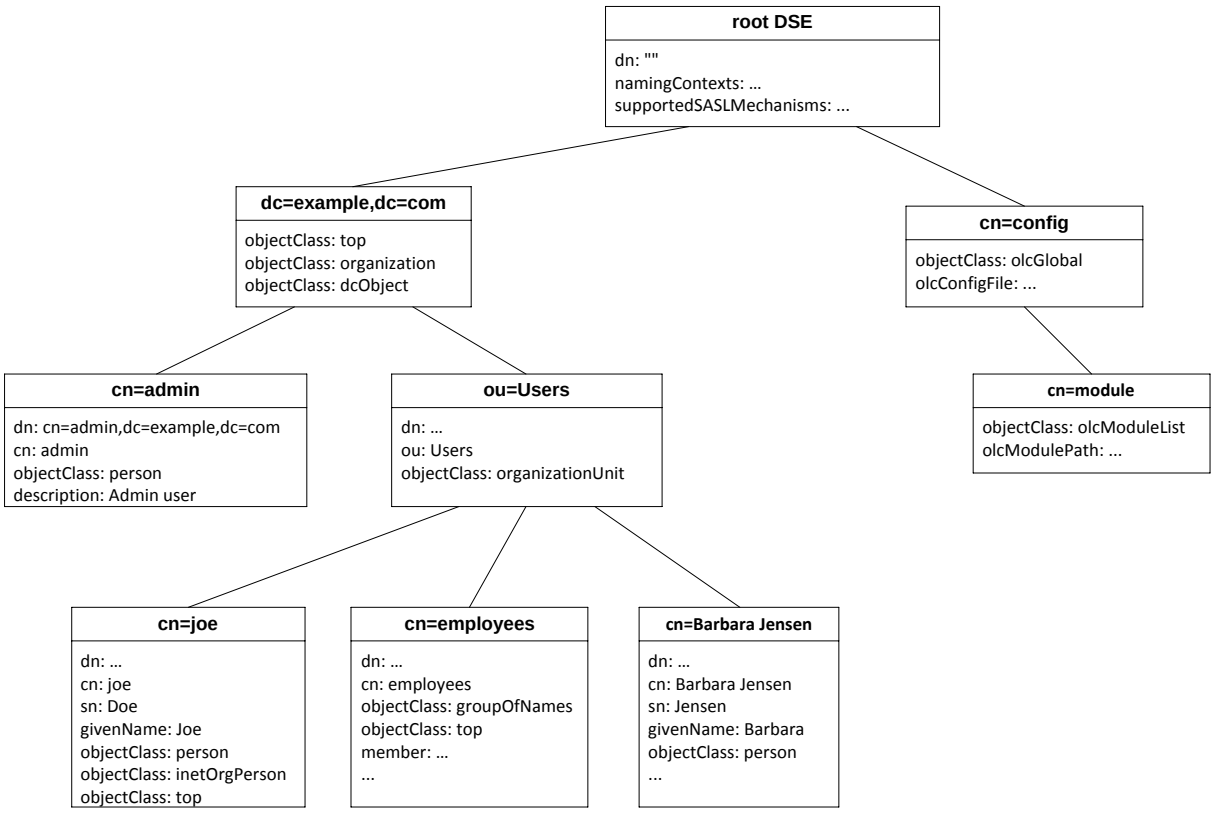


Figure 2. Example LDAP directory

Observe the followings:

- The directory is a hierarchical *tree* structure, the nodes are called *entries*.
- The information is stored in sub trees called naming contexts, in the above example there are two naming contexts defined: `dc=ecample,dc=com` and `cn=config`.
- Every entry has attributes (attribute names + values).
- The *type of* an entry is defined by its *objectClass* attribute (this is the typeOf relation from the modeling topic in the beginning of the course).
    - LDAP supports multi-valued attributes, observe that the `dc=ecample,dc=com` entry has 3 values for its objetClass attribute.
    - The values of the objectClass attribute refer to classes defined in the *schema*.
- As with any tree structure (for example XML documents), the following sets can be defined for any entry: ancestors / children / parent / siblings.
- The entries are identified by:
    - RDN (relative distinguished name): the name of the attribute, which uniquely differentiates the entry from its siblings. E.g.: for the entry `joe` the RDN is `cn`.
    - DN (distinguished name): the concatenation of the entries and its ancestors RDNs. E.g.: `cn=joe,ou=Users,dc=example,dc=com`

4.3 Open source implementations for Linux

There are several LDAP directory implementations available, e.g. OpenLDAP or Apache Directory Studio. In the home assignments we will work with OpenLDAP.

The following software will be useful.

- OpenLDAP: this is the directory server storing the information.
- phpLDAPadmin: this is a web-based GUI for the directory.
- Command line tools (ldapsearch, ldapmodify): these command line tools can be used to access the directory servers from scripts.

The following figure displays the login screen of a phpLDAPadmin instance.

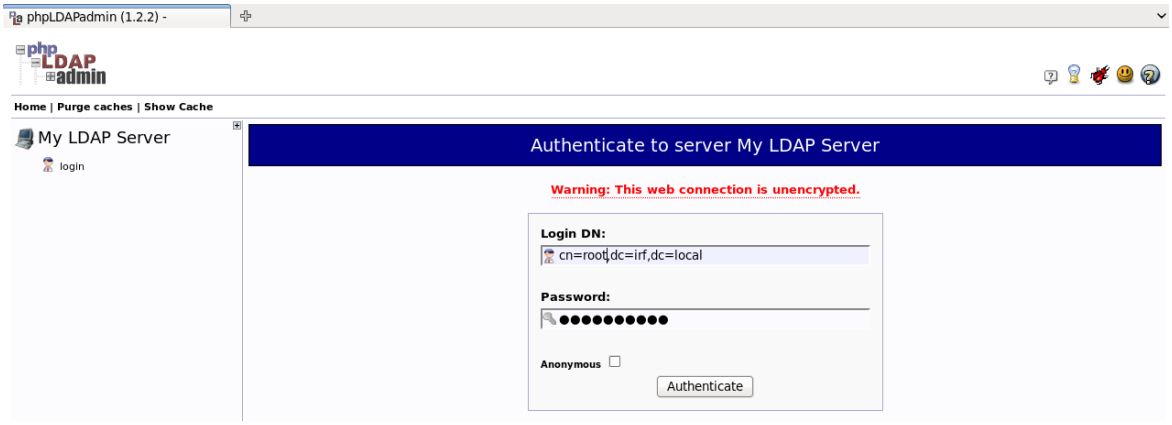


Figure 3. Login screen for phpLDAPadmin

Once we login, the directory structure is displayed.

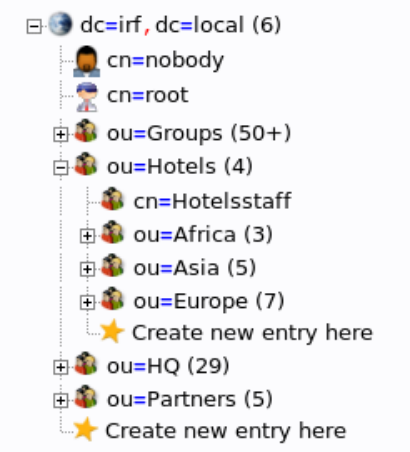


Figure 4. Example directory structure

A virtual machine will be provided on the course's web site that contains an OpenLDAP with example data. Use this virtual machine to familiarize yourself with LDAP.

The following exercises are recommended:

- View the details of a user and a group.
- Try to modify an attribute of a user (e.g. try to change its display name).
- Ty to create a new user and add it to an existing group.
- Try to search for those users, whose surname starts with 'b'.

Using the command line tools

Start with the ldapsearch tool, which displays information from the directory. An example for a non-trivial query:

```
ldapsearch -H ldap://localhost:389 -x -b "ou=HQ,dc=irf,dc=local" -s one "(&(objectclass=person)(sn=b*))" cn sn
```

The command is built up in the following way:

```
ldapsearch <switches> <filter expression> <list of attributes to retrieve>
```

Important switches are:

- -H: location of the LDAP server to contact (protocol://hostname:port format),
- -x: use simple authentication,
- -b: base of the search,
- -s: type of search (one: only one level).

The filter should be specified in a prefix format.

Exercise: Try to query those groups that are located under the Hotels organizational unit and start with an 's' character.

If in doubt, have a look at Section 14 of [8]

5 Configuration management

This topic will be about collecting information about the different components in a large, heterogeneous IT system. The motivation is the following. If we have a complex infrastructure, with several servers, switches and routers, different operating systems, numerous services and applications, then having an up-to-date inventory of all the settings and states of the elements is crucial. Usually the inventory is collected for different aspects of the system (e.g. only for network settings), resulting in so-called information silos. To overcome this issue a common model and database is needed that can store all the relevant information from the whole hardware and software stack.

5.1 Reading material

[10] Vitezslav Crhonek. "WBEM/CIM Management – Basic concepts and availability in Fedora", 2011, URL: http://vcrhonek.fedorapeople.org/wbem.pp.pdf

This is a good introduction slide show about the basic WBEM/CIM concepts.

[11] Praveen Kumar Paladugu. "WBEM Based Management in Linux", Dell Tech Paper, 2011, URL: http://linux.dell.com/files/whitepapers/WBEM_based_management_in_Linux.pdf

This is a good overview about the different WBEM technologies and tools.

Additional reading (optional):

[12] CIM, WBEM specifications of the DMTF, URL: http://www.dmtf.org/

5.2 Specifications and technologies

Figure 5 presents a general architecture for a configuration management solution.

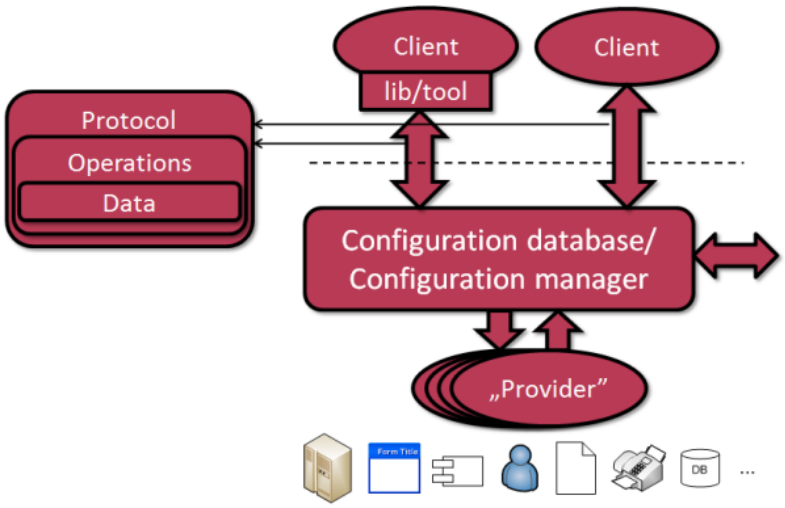


Figure 5. General architecture for configuration management

It has the following elements (bottom to top):

- *Managed elements*: hardware and software elements of the system that needs to be managed.
- *Providers*: providers are software components that can query the managed elements and collect information from them or can invoke actions on the elements.
- *Configuration database*: special database to store configuration information. The database should provide a standardized import/export interface.

- *Remote access interface*: the database should provide an interface to access its information remotely. It should specify the protocol to use, the possible operations to offer and the representation of the configuration data.

The topic will introduce the solution proposed by the *Distributed Management Task Force* (DMTF) consortium. DMTF is a group of large vendors and organizations (e.g. AMD, Cisco, HP, IBM, Intel, Microsoft, Oracle, RedHat, VMware…) that create specifications to enable interoperable IT management. Their approach was to specify a common model, a configuration database and protocols to access this database from different platforms.

Figure 6 presents the main specifications developed by the DMTF.

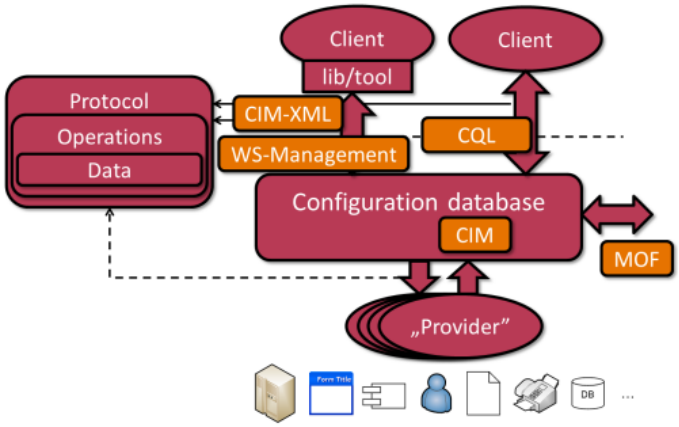


Figure 6. Configuration management specifications of the DMTF

- *Common Information Model (CIM)*: CIM is a large model for representing IT elements in a vendor and platform-independent manner. It consists of the following parts.
    - *CIM Metamodel:* defines a modeling language to represent CIM models. It is specified using the UML metamodel, but has slightly different concepts.
    - *CIM Schema:* the configuration management model specified by the DMTF. The model can be extended by vendors.
    - *Managed Object Format (MOF):* a textual import/export format to represent CIM models. The classes of the CIM Schema are also specified in MOF format.
    - *CIM Object Manager (CIMOM):* a CIM compliant configuration database is called a CIMOM. The specification does not prescribe its internals, it should only be able to import and export MOF models.
- *Web Based Enterprise Management* (WBEM): a collection of specifications defining the remote access interface of a CIMOM:
    - *CIM-XML:* an HTTP-based protocol that encapsulates the description of CIM classes and instances in XML format, and defines the possible operations against a CIMOM.
    - *WS-Management*: a protocol defined over SOAP-based Web Services. WS-Management is a general remote management protocol, but has the necessary bindings defined to represent CIM data.
    - *CIM Query Language (CQL)*: a SQL-like query protocol to filter the data of a CIMOM.

5.2.1 Common Information Model

The CIM Schema is a large collection of models (the actual version defines more than 1400 classes!). Figure 7 depicts a small fragment of the model. The notation should be familiar from

UML. Classes can have typed properties and methods. Relations can be associations, composition/aggregation or generalization. The multiplicities are defined in the usual way.

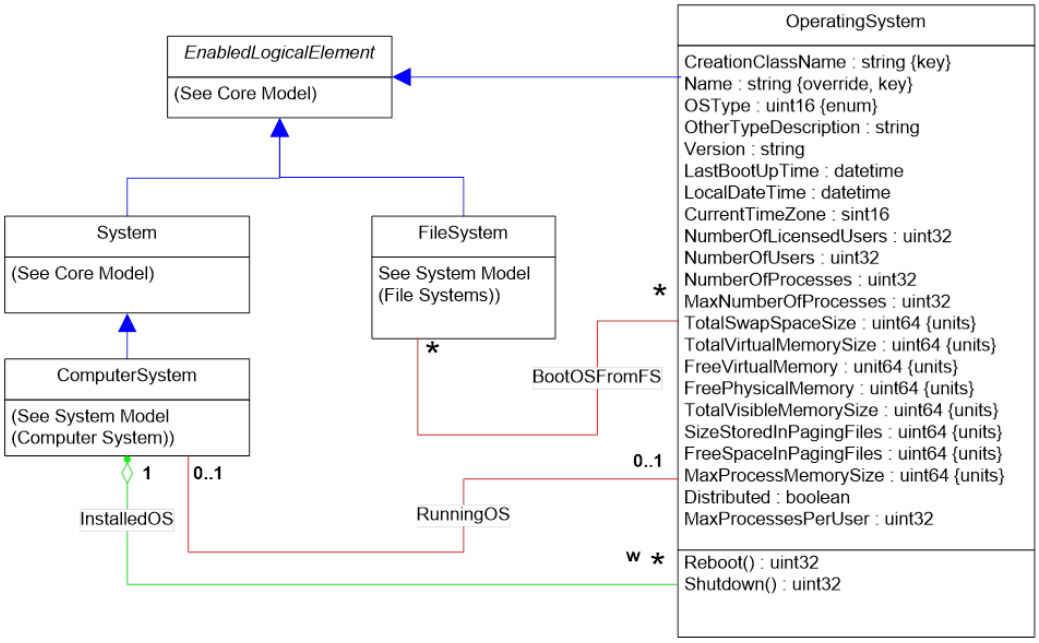


Figure 7. Fragment of the CIM Schema

Model elements can be represented in the textual MOF format. This format is similar to interface definition languages; the syntax resembles C/C++. The following is a short excerpt from a MOF definition.

```
[Description ("The Location class specifies the position of a PhysicalElement." )]
class CIM_Location : CIM_ManagedElement {
   [Key, MaxLen ( 256 )]
   string Name;

   [Key] string PhysicalPosition;

   string Address;
};
```

The [] symbol represents qualifiers. They add meta-information to a definition of a class (like stereotypes in UML). For example, this class has the *Description* qualifier. The example defines the *CIM_Location* class, which is a subclass of *CIM_ManagedElement*. The name of a class is in the format of `Schema_Class`, where the schema is like a namespace in programming languages; class names in a schema should be unique. The *CIM_Location* class has three properties and no methods. An instance of a class is uniquely identified by the values of its key properties. The CIM_Location class has a compound key consisting of the *Name* and *PyhsicalPosition* properties.

5.2.2 Web Based Enterprise Management

From the WBEM specifications we will take a closer look at the CIM-XML protocol. The CIM-XML uses the HTTP protocol to make requests and receive answers. It defines several operations over a CIMOM; the following table lists some of the common ones.

A GET operation gets a specific class or instance (a class is identified by its name; an instance by its class name and the values of its keys). If all instances of a class should be accessed, then an ENUMERATE operation should be used. For both classes and instances there exist operations that only retrieve the name and not all the details. A CQL query can be executed with the EXECQUERY operation. Finally, the associations between classes can be managed by the

association operations. The ASSOCIATORS operation gets for a given instance all the other instances that are connected to the given instance.

Table 1. Common CIM-XML operations

Type	Operation
Class	GetClass EnumerateClasses EnumerateClassNames
Instance	GetInstance EnumerateInstances EnumerateInstanceNames GetProperty
Association	Associators AssociatorNames References
Query	ExecQuery

In a CIM-XML request, classes and instances are identified by their object path. It is in the format:

```
<protocol>://<namespace path>:<class name>[.<property=value[…]>]
```

The namespace path is implementation-specific, but it usually contains the IP address or hostname, the port of the CIMOM and the CIM namespace to use. For example, this is a valid object path:

```
http://192.168.1.10:5988/root/cimv2:CIM_Processor.DeviceID=0
```

The above path describes that the IP address of the CIMOM is 192.168.1.10; it should be contacted on the 5988 port (which is the standard port for CIM-XML); in the CIMOM the root/cimv2 namespace should be used (which is the commonly used namespace for standard CIM classes); and the request is about that instance of the *CIM_Processor* class, which has a *DeviceID* property value of 0.

5.3 CIM implementations and tools

There are several commercial or open source implementations for the CIM and DMTF specifications. In the course, we will use the following open source software running on Linux.

- sblim-sfcb[2]: a CIMOM implementation.
- wbemcli[3]: a command line client for accessing CIMOMs using the CIM-XML protocol.
- yawn: a small web based interface to depict the content of the CIMOM.

A new version of the virtual machine will be published on the course web site that will have these tools installed and configured. Perform the following exercises:

- Explore the contents of the CIMOM using YAWN (http://localhost/yawn in the VM). Try to look at the properties of some of the CIM classes, try to enumerate instances.
- Try out the *wbemcli* tool. Its manual and the texts in the reading material section offer several examples. A simple request looks like:

```
wbemcli -nl gc 'http://meres:LaborImage@localhost:5988/root/cimv2:CIM_OperatingSystem'
```

[2] http://sourceforge.net/apps/mediawiki/sblim/index.php?title=Sfcb
[3] http://sourceforge.net/apps/mediawiki/sblim/index.php?title=Wbemcli

6 Virtualization and cloud computing

Virtualization is broad concept that is used everywhere in software engineering. Operating systems provide virtual memory and virtual file systems, C++ uses virtual functions, Java offers a specific virtual machine. This topic will be about a platform virtualization (also known as server virtualization), a techniques that makes it possible to create and run virtual machines on a physical computer

6.1 Reading material

[13] Zoltán Micskei. "Virtualization", lecture notes for the Operating systems course, URL: http://mit.bme.hu/~micskeiz/opre/operating-systems.html

Refresh the virtualization materials from the operating systems course.

[14] NIST. "The NIST Definition of Cloud Computing", SP 800-145, Sept. 2011, URL: http://csrc.nist.gov/publications/PubsSPs.html#800-145

This is a commonly accepted definition of cloud computing terms.

[15] Jinesh Varia, Sajee Mathew. "Overview of Amazon Web Services", Amazon whitepaper, October 2012, URL: http://media.amazonwebservices.com/AWS_Overview.pdf

This document introduces the main components of AWS, a major cloud computing provider.

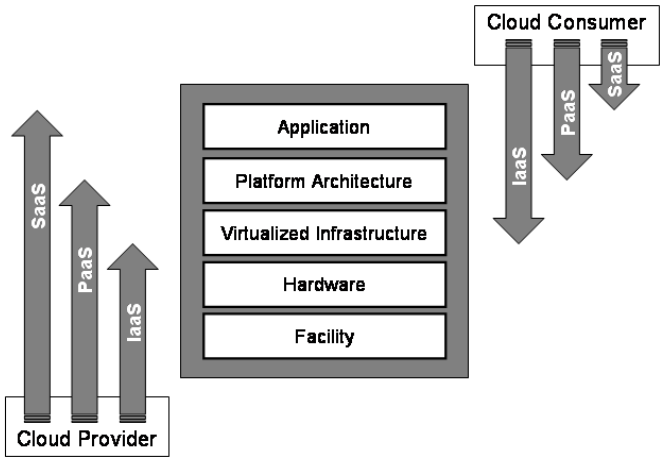
Additional material (optional):

[16] NIST. Cloud Computing Synopsis and Recommendations, SP 800-146, May 2012, URL: http://csrc.nist.gov/publications/PubsSPs.html#800-146

6.2 Concepts and technologies

After reading the materials and preparing on the current topic, the students should understand the following concepts and technologies:

- virtualization, platform virtualization, difference between virtualization and emulation, virtualization techniques (software, para-virtualization, hardware-assisted) [these were covered in the operating systems course]
- cloud computing term, examples for cloud computing services, essential characteristics of cloud computing, service models, deployment models.



8. Figure: Cloud deployment models [source: NIST]