maXbox Starter 29

Start with UML

1.1 A Model Language

The purpose of building programs is to fulfil a need. And a program needs structure with functions. UML can help to structure the system and find the functions.
UML is said to address the modeling of manual, as well as systems, and to be capable of describing current systems and specifying new ones. UML (Unified Modeling Language) is intended to be an analysis and design language, not a full software development methodology. It specifies notations and diagrams, but makes no suggestions regarding how these should be used in a development process.

A first step in UML is to find the requirements.
Requirements are statements of what the system must do, how it must behave, the properties it must exhibit, the qualities it must possess or fulfill, and the constraints that the system and its development must satisfy.

This is how we declare a function we want to use from a sort of Calendar Library:

```
Function OpenCalendar(dwDesiredAccess: DWORD; bInheritHandle: BOOL;
                      dwProcessId:DWORD): THandle;
   External 'OpenCalendar@gdi32.dll stdcall';
```

☞ As you remark, this is NOT a requirement, it's a solution.

& Yogi Berra said: "You've got to be careful if you don't know where you're going 'cause you might not get there!"

eT Stephen Covey said: "Seek first to understand."

Both statements are applicable to software development as well as many other areas!

Both statements should be kept in mind as you write your requirements to help you make sure your requirements will assist you in delivering a software solution that meets your users' needs. You can find all sorts of templates and formal processes for requirements of various kinds, and while they are useful, the biggest problem I've found is that most people confuse defining the need with proposing a predefined solution.

As soon as a use case driven requirements document contains any part of "how we're solving this", you've crossed the line into presupposing that you already know what the problem is and can stop listening.
Suppose you want to use a function of the class `TDateTimePicker` just to pick a date or navigate at the calendar is just presupposing.
All you have to do is to make sure your problem description does not include any part of a technical solution or you are blinding yourselves to alternatives and/or assuming you know more than you probably do.
Years ago there was the idea of executable UML.
Executable, translatable UML (xtUML) is an extension to UML, based on the Shlaer-Mellor method of model-driven architecture, which supports a powerful approach to model-driven development and using a predefined library like a template.
A use case library can be used by several applications at the same time. Some libraries are provided with the UML system and available for any application. Other libraries are written for a particular framework and are loaded within the application[1].

```
ProcessHandle:= OpenCalendar(PROCESS_QUERY_INFORMATION or
                             PROCESS_VM_READ, false, GetCurrentDate);
```

The const `PROCESS_QUERY_INFORMATION` retrieves certain information about a process, such as its token, exit code, and priority class and the other const `PROCESS_VM_READ` is required to read memory in a process using. By the way the script you get all this is:

http://www.softwareschule.ch/examples/461_sqlform_calwin.txt

1.2 Use Case

Use-case diagrams, which model the functional requirements of the system in terms of actors and actions (e.g., customer gives cheque for cashing to counter clerk or user picks a date), are the top most approach to find requirements. A design follows requirements in analysis therefore form follows function. So producing an application consists of implementing all the use-cases in a model.

Experts agree, Use Cases are the most effective means of capturing and documenting functional requirements.
If you use Use Cases or are planning to use it to document your system's behaviour, the notation will help improve your understanding.
With our date picker that just copied and pasted from the script folder, we will use a next example to retrieve calendar information. Until this call here some more topics.

Make sure your requirements include things that are necessary. Don't add things because they're "easy" or "cheap". There have been many, many unsuccessful products that had a laundry list of features implemented that never satisfied their market.

The advantage of a UML is:
For most applications, use cases, classes and packages in an object oriented sense provide greater flexibility and are easier to create than programming without modelling. However, there are several situations where use cases would be better suited to your projects than packages, but anyway here the advantages of UML:

- Provides standard for software development.
- Reducing of costs to develop diagrams of UML using supporting tools.
- Development time is reduced.

[1] PSM Platform Specific Model

- The past faced issues by the developers are no longer exists.
- Has large visual elements to construct and easy to follow. Your code module will be called from non technical persons.
- You are extending the functionality of a requirement.
- You are creating a code module to be used by third-party developers.
- Your project is business- and use case driven.

UML shows the future modelling where the entire applications are generated from high-level UML models and highlights the best practices for adopting UML in an enterprise.
I don't explain those points for your insights. Let me say a few words about building a UML driven function or procedure.

This code solution lies behind the following use case diagram.

```
procedure TdbOpenDlg_CalBtnFromClick(Sender: TObject);
begin
  dbOpenDlg:= TdbOpenDlg.Create(self)
  with dbOpenDlg do begin
    setbounds(0,0,400,400)
    with TImage.Create(dbopendlg) do begin
      parent:= dbopenDlg;
      Setbounds(150,20,100,100)
      picture.bitmap:= getbitmap(Exepath+'examples\images\time.bmp')
    end;
    //showmodal goes later
  end;
  with TBitBtn.Create(dbopenDlg) do begin
    parent:= dbopenDlg;
    setbounds(20,300,150,40)
    caption:= 'OK Date';
    glyph.LoadFromResourceName(getHINSTANCE,'TPSIMPORT_COMOBJ');
    onClick:= @TdbOpenDlg_Button1Click;
  end;
  try
    with TDateTimePicker.Create(dbopenDlg) do begin
      parent:= dbopenDlg;
      SetBounds(20,20,100,100);
      if dbopenDlg.showModal = mrOK then
        beginDate:= Datetime;
      writeln('date with time: '+datetimetoStr(Datetime));
    end;
    writeln('get date back '+datetoStr(begindate));    //debug
  finally
    dbopenDlg.Release;     //than free
  end;
end;
```

Therefore we use in maXbox also a modal form but in the runtime constructor loading of the script engine with a modal result as the date are picked.

```
procedure TdbOpenDlg_Button1Click(Sender: TObject);
begin
  dbOpenDlg.ModalResult:= mrOK;
end;
```

You can use also a try finally handling in the call and set the object free with release. For best results, exceptions should be caught within the scope of the form throwing the exception for example if the image `picture.bitmap` couldn't found.

O.k. enough of technical grabbing let's chose and explain the second function (procedure) called `TdbOpenDlg_Button1Click()` to glue it with the first function, hope you find it above, `TdbOpenDlg_CalBtnFromClick()` together.

The magic is the event handler of the bit button:

```
  with TBitBtn.Create(dbopenDlg) do begin
    parent:= dbopenDlg;
    setbounds(20,300,150,40)
    caption:= 'OK Date';
    glyph.LoadFromResourceName(getHINSTANCE,'TPSIMPORT_COMOBJ');
    onClick:= @TdbOpenDlg_Button1Click;
  end;
```

Now the dialog itself is just an alias to a form class:

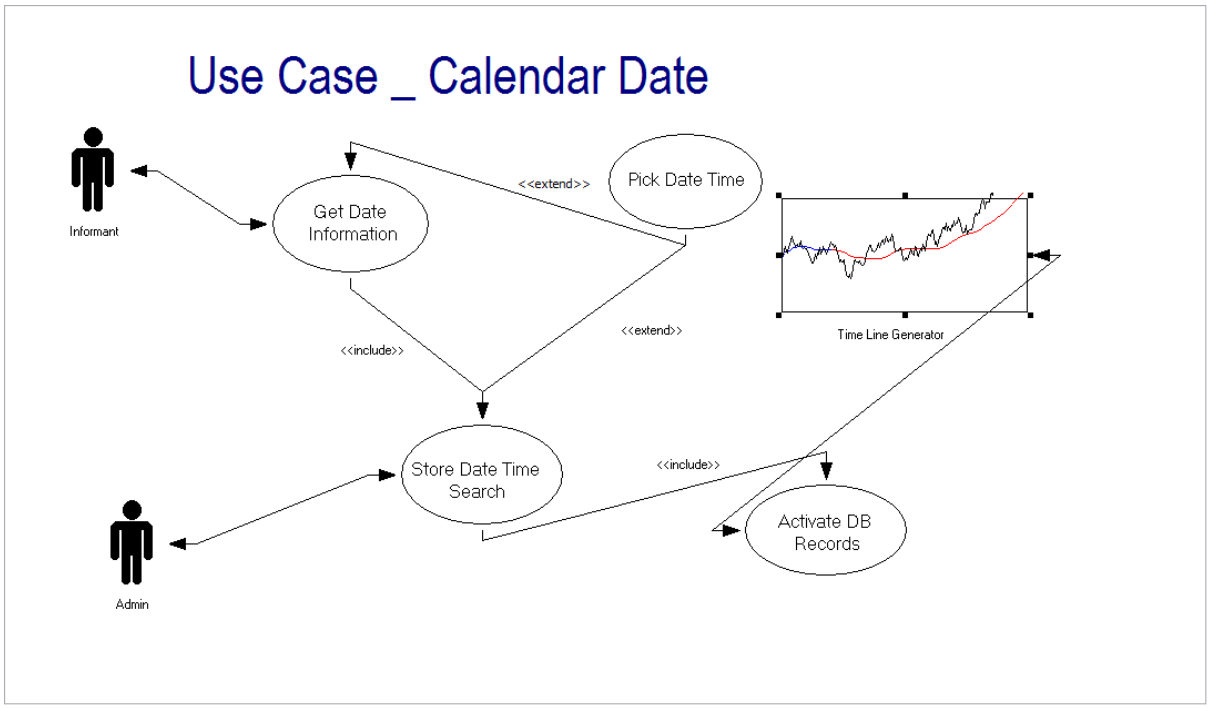
```
type TdbOpenDlg = TForm;

var
  dbOpenDlg: TdbOpenDlg; //Alias;
     External 'GetProcessMemoryInfo@psapi.dll stdcall';
```

Interesting point is the function `GetProcessMemoryInfo` which we use came from another DLL called `psapi.dll`. Now we switch to the Use Case Modeller:
You find the model form editor of the app in menu

`../Program/Use Case…//Add Use Case`

To understand the UML model, one need not know detailed technical knowledge. With this tool you can open a second frame or form and explore which functions are inside.



A use case is a written description of how users will perform tasks on your application. In our example a user wants to get calendar information and set a specific date. From a user's point of view, a system's behaviour as it responds to a request.

When you save the diagram (model) you do have a relation to your script code:

```
461_sqlform_calwin.txt
461_sqlform_calwin.uc
```

You find a lot of discussion concerning this set of model and code. Model first or contract first is one side of the medal and the other means planning.
Project planning and tracking is a part of project management. When the requirements are stated and agreed upon by the client, a plan needs to be made....

1.2.1 Elements of a Use Case

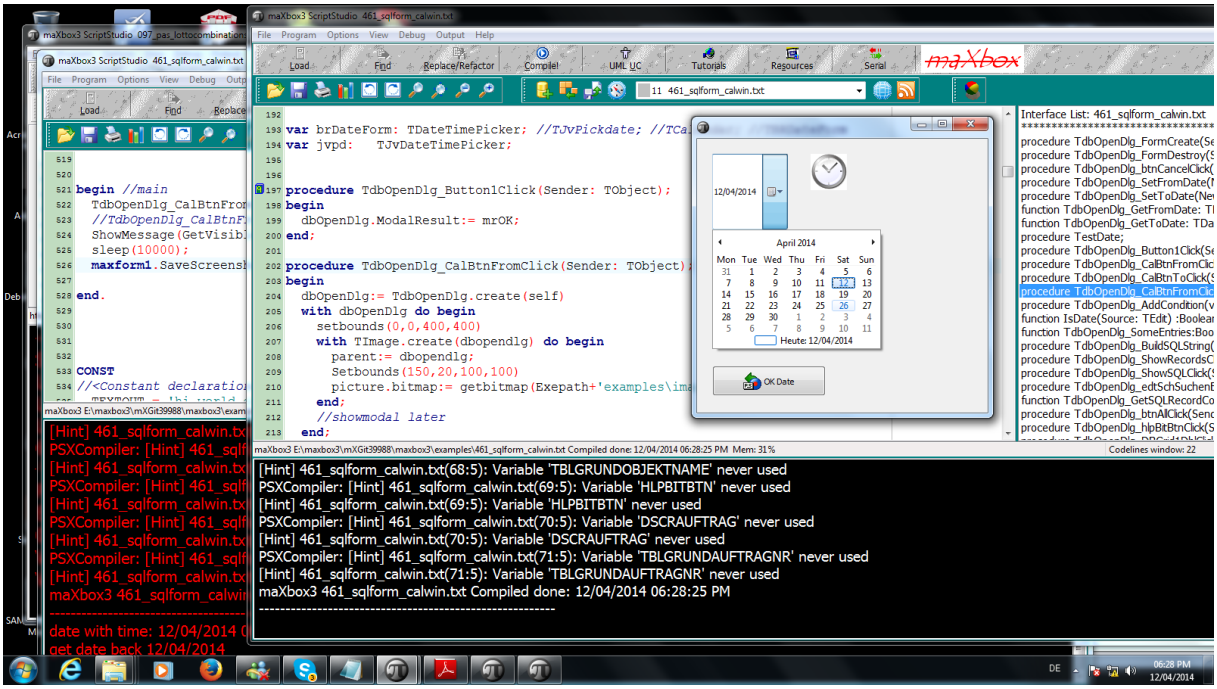
Depending on how in depth and complex you want or need to get, use cases describe a combination of the following elements:

- Actor - anyone or anything that performs a behaviour (who is using the system)
- Stakeholder - someone or something with vested interests in the behaviour of the system under discussion
- Primary Actor - stakeholder who initiates an interaction with the system to achieve a goal
- Preconditions - what must be true or happen before and after the use case runs.
- Triggers - this is the event that causes the use case to be initiated.
- Main success scenarios [Basic Flow] - use case in which nothing goes wrong.
- Alternative paths [Alternative Flow] - these paths are a variation on the main theme. These exceptions are what happen when things go wrong at the system level.

Answer: It is reliable, but you have to very careful when interpreting the results.

☞The requirement dynamics of a process is subject to lots of cases. A quick rundown might be:

- how much precondition and triggers are there
- how much context switching ids going on
- how many processes are running
- are there processes that need a lot of time or less time but a high frequency
- Does your process running asynchronous or not…...



1.2.2 Calendar Package

Above you see the Use Case at work, maybe a DLL, component or a package. A DLL in windows is stored in a binary PE format and could be a UML package with a namespace. This prevents a software- and hardware-independent way of accessing functions or storing data.
We do have a look at the Package or Component diagram below.
The file formats for DLLs are the same as for Win EXE files – that is called, Portable Executable (PE). Another topic is maintenance. In most cases, when you get a message that a DLL file is missing it is because a program has been installed or uninstalled incorrectly.

> In any case, if the DLL is associated with a program you are using and want to go on using, in most cases just reinstall the program. That should restore the file.

Let's go back to the `OpenCalendar()`. We use for most of your configuration an XML file to describe the build or install process with the necessary DLL files mentioned. Suppose the calendar is a component and needs a database. This package config file for example has the purpose of an ini-file:
http://max.kleiner.com/myconfig.xml

The ini-file format is still popular; many configuration files (such as desktop or persistence settings file) are in this format. This format is especially useful in cross-platform applications and mobile devices where you can't always count on a system registry for storing configuration data or save the user options.
It shows the implementation of 2 databases (`Interbase` and `mySQL`) and the needed DLL which communicate with a client over `TCP` and sockets. I made it scriptable to demonstrate the fascination of 2 database boxes communicate with each other at the same time.

```
<?xml version="1.0" ?>
- <databases>
   - <database db="Firebird23">
      - <connection>
          <name>TCalendarTrueSoft</name>
          <path>\\testserver\SuperPath</path>
          <dll >GDS32.DLL</user>
        </connection>
      </database>
   - <database db="mySQL55">
      - <connection>
          <name>maXSQLDates</name>
          <server>/.//kylix05/mXbase</server>
          <dll>mySQL.DLL</user>
        </connection>
    </database>
  </databases>
```

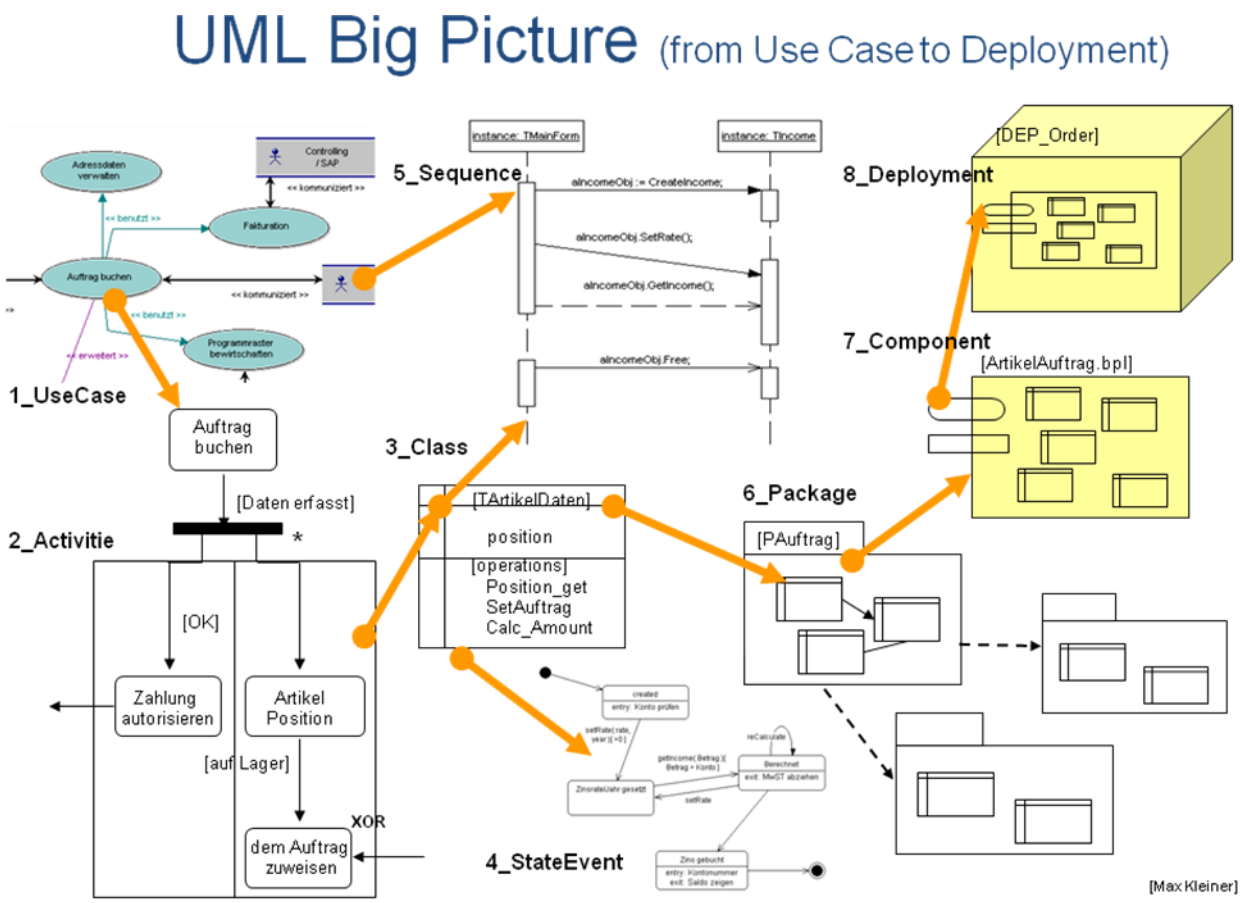
`ExePath()` is a useful function where you always get the path of maXbox.
If someone tries to start (install) the script or the app to or from a different drive for space or organizational reasons, it may fail to (install) or to run the script after installation[2].
You find all info concerning run environment of the app, DLL and script in menu

```
../Program/Information/……
../Debug/Modules Count/…… {List of all loaded DLLs!}
../Debug/Units Explorer/  {Packages}
```

[2] You don't have to install maXbox, just unzip and drop a script

Next we go to the big picture. It shows the traceability from a use case over a package to the deployment diagram. Requirements can be dependent on other requirements and other project artifacts, such as blueprints, flow charts, business rules and test cases.
These dependencies create a network of links which need to be managed throughout the whole lifecycle of a project.

☞ Traces can be of different types and are often managed via traceability matrixes.
Nevertheless, we will refer to it as a V method because this is the standard terminology for our CASE tools and consulting service. So we are able to stick together all diagrams in big picture phase logic with the following traceable examples:
So you see now the UML Big Picture which shows the source of UML traceability:



3. Traceability of UML

Many system DLLs or namespace packages are called key system components (`kernel32.dll`, `advapi32.dll`, `ntdll.dll`, `hal.dll`, `user32.dll`, `gdi32.dll`, and so forth). If you have ever programmed in Pascal or C, then you would know that unlike any object-oriented programming language like Object Pascal, the basic unit of organization is the function; that C or Pascal is a function, or procedural (not object) oriented language.

You can always download the current official version of UML and its associated specifications from a specifications catalogue page for modelling and metadata Specifications at

http://www.omg.org/technology/documents/modeling_spec_catalog.htm

If you're a modeller, especially if you're just starting out, you will probably find the specifications themselves a bit hard to read.

✌Keep in mind that at omg.org they're the formal definition of the modelling language itself, and not an instruction book like the V-Model.

That’s the price for formalism at runtime.
Such a standard way of describing process data would enable a user to send an intelligent agent (a program) to each computer maker's web site, gather data, and then make a valid comparison. A UML package or component can be used by software or companies that want to share information in a consistent way.

☞ Libraries can be loaded at runtime and thus shared between different concurrent applications. This is called dynamic linking.

1.3 Another Sequence Traceability

In this short last part I will give an overview of the diagrams:

1. Use Case shows in the simplest form the graphical representation of a user's interaction with a system.

2. Activity diagrams show the activities of a particular operation in the system. The activity diagram consists of an initial mode;

3. Class diagrams shows the classes used in the system. Each class has some properties and operations.

4. State-Event diagrams are used to show the states that an object can occupy, together with the actions which will cause transitions between those states.

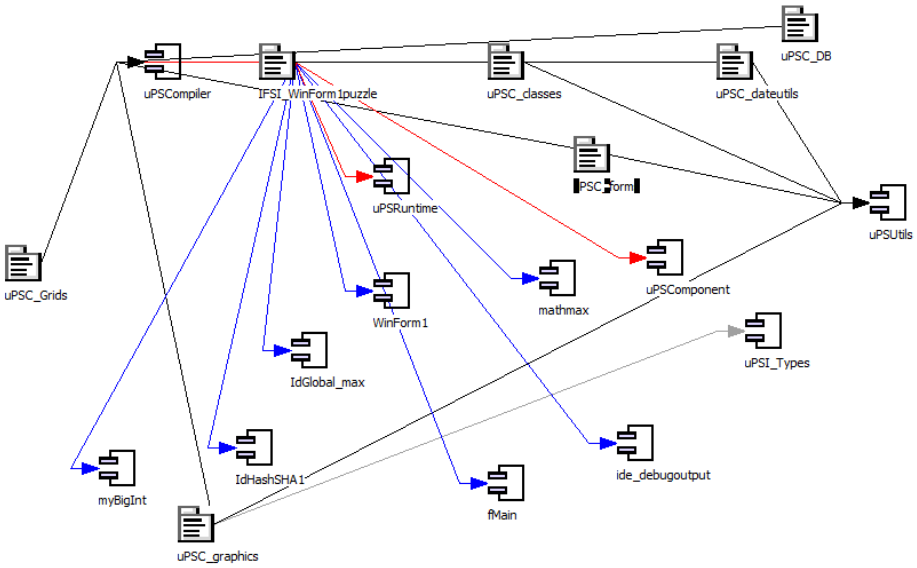
5. Sequence diagrams show how objects interact through time, by displaying operations against a timeline.

6. Packages are organizational units of namespaces from the source code

7. Components are used to describe parts of a system at a high level, such as “report generator”. These will have internal structure, and will allow you to see a detailed view of its structure.

8. Deployment diagrams are used to show how the model relates to hardware and, in a multiprocessor design, how processes are allocated to processors.

Now it’s time to put those pieces together and serve the highlight. The Time Machine!



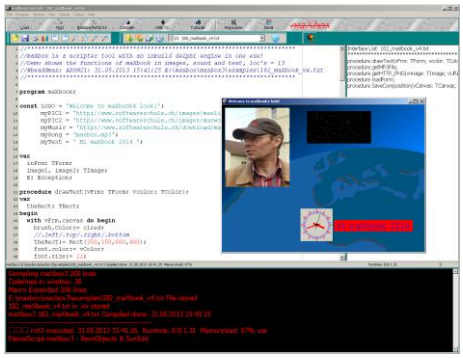
4. Packages in Dependencies

A package in the UML is used to group elements, and to provide a namespace for the grouped elements. A package may contain other packages, thus providing a hierarchical

organization.

Links of maXbox and UML:

http://www.softwareschule.ch/maxbox.htm
http://sourceforge.net/projects/maxbox
http://sourceforge.net/apps/mediawiki/maxbox/
https://github.com/maxkleiner/maXbox3.git
http://max.kleiner.com/uml2.htm
http://www.softwareschule.ch/download/patternposter.pdf
http://www.sprez.com/articles/requirements-solutions.html



1.4 Appendix I Package

```
{*******************************************************}
{       Borland Delphi Run-time Library                 }
{       WinNT process API Interface Unit                }
{                                                       }
{       Copyright (c) 1985-1999, Microsoft Corporation  }
{       Translator: Borland Software Corporation        }
{*******************************************************}
unit PsAPI;
interface
{$WEAKPACKAGEUNIT}
uses Windows;
{$HPPEMIT '#include <psapi.h>'}
type
  PPointer = ^Pointer;
  TEnumProcesses = function (lpidProcess: LPDWORD; cb: DWORD; var cbNeeded:
DWORD): BOOL stdcall;
  TEnumProcessModules = function (hProcess: THandle; lphModule: LPDWORD;
cb: DWORD;
    var lpcbNeeded: DWORD): BOOL stdcall;
  TGetModuleBaseNameA = function (hProcess: THandle; hModule: HMODULE;
    lpBaseName: PAnsiChar; nSize: DWORD): DWORD stdcall;
  TGetModuleBaseNameW = function (hProcess: THandle; hModule: HMODULE;
    lpBaseName: PWideChar; nSize: DWORD): DWORD stdcall;
  TGetModuleBaseName = TGetModuleBaseNameA;
  TGetModuleFileNameExA = function (hProcess: THandle; hModule: HMODULE;
    lpFilename: PAnsiChar; nSize: DWORD): DWORD stdcall;
  TGetModuleFileNameExW = function (hProcess: THandle; hModule: HMODULE;
    lpFilename: PWideChar; nSize: DWORD): DWORD stdcall;
  TGetModuleFileNameEx = TGetModuleFileNameExA;
  {$EXTERNALSYM _MODULEINFO}
  _MODULEINFO = packed record
    lpBaseOfDll: Pointer;
    SizeOfImage: DWORD;
    EntryPoint: Pointer;
  end;

  {$EXTERNALSYM MODULEINFO}
  MODULEINFO = _MODULEINFO;
  {$EXTERNALSYM LPMODULEINFO}
  LPMODULEINFO = ^_MODULEINFO;
  TModuleInfo = _MODULEINFO;
  PModuleInfo = LPMODULEINFO;

  TGetModuleInformation = function (hProcess: THandle; hModule: HMODULE;
    lpmodinfo: LPMODULEINFO; cb: DWORD): BOOL stdcall;
  TEmptyWorkingSet = function (hProcess: THandle): BOOL stdcall;
  TQueryWorkingSet = function (hProcess: THandle; pv: Pointer; cb: DWORD):
BOOL
```