Developing for the Agenda: A case study

Michaël Van Canneyt

March 16, 2014

Abstract

In this article, we show how to develop applications for the Agenda VR3 from Agenda Computing. This is done by presenting a case study for a small application which was developed as a test for the usability of the Agenda in a school environment.

1 Introduction

In this article some guidelines for developing applications for the Agenda are presented, and an evaluation The application that was developed is part of a suite of Administrative applications for schools. In order to keep track of which students are present in school, absentees are listed twice a day. In the usual case, this means that some administrative aid does a tour of the school building, and collects at each classroom a small papers with the names of pupils that are absent. Upon return in the administrative office, the names on the papers are entered in the computer and stored in a database. Pupils that are absent too often, are reported to the ministry of education.

In an attempt to smoothen the task of collecting absentees, a small application was implemented, which should run on an Agenda. The idea is that before collecting the absentees two actions occur:

1. A list of classrooms is loaded in a particular order. For each classroom, a list of students which should be present is loaded. Multiple configurations can be loaded into the agenda.
2. A list of known absentees is loaded into the agenda.

The first action can occur once a year, when new classes are formed. In principle, it must be repeated whenever the list of classrooms or pupils changes. The second action can be repeated every day before the administrative aid starts his/her tour of the classrooms. It does not have to be done, and serves only to make the task easier by already marking the student as absent (the student can be marked as present if needed)

When the tour is finished, the Agenda is again synchronised and the list of absentees is loaded into the computer containing the administrative database. The data transfer (both from and to the Agenda) is done by means of simple text files in a Windows .ini format.

The agenda application consists of 2 screens. The first screen presents a choice of configurations. Each configuration represents an ordering of the classrooms in the order they are visited by the administrative aid. Or, in a system where the pupils change classrooms, it represents the situation at the hour when absences are taken.

Once the configuration is chosen, a list of classrooms is presented. Choosing a classroom, and pressing a button, a second window with the pupils that are expected to be present in the classroom is shown (the list of pupils is loaded from the configuration files). This window consists of a scrollable area, filled with checkboxes, one per pupil. When the pupil is absent, his or her corresponding checkbox is checked. Pupils that are known to be absent are already checked.

When the second window is closed, the list of absent pupils is saved to a file.

There is an option to automatically open a new window with the list of pupils of the next classroom. If the classrooms are correctly ordered, then the correct list of pupils is already on the screen by the time the administrative aid reaches the next classroom.

This application could be developed on any PDA, but the Agenda has the advantage of being cheap when compared to other PDAs; Schools often have limited budgets so cost is often a decisive factor.

2 The Agenda Development Environment

The Agenda runs Linux - as such, developing an application for the Agenda is no different from developing for Linux or for most unices. However, there are some small caveats:

1. The agenda does not (yet) run a compiler - all compilation must be done as cross-compilation. By itself, this is not a real problem, but it does means some extra work is needed.
2. The agenda has limited resources: A small screen, and little memory, even power consumption is also a factor not to be taken lightly. This mainly affects the GUI toolkits that are available: Most toolkits use lots of system resources (i.e. mainly memory and CPU speed) and so can be considered not really suitable for the Agenda. It would for instance not be a problem to run GTK or even Qt on the agenda, but the system would probably not be capable of running two such applications at the same time.

When developing for the Agenda, one should also be prepared to spend some time on browsing and searching the web. While there is plenty of information on the Agenda, it takes some time to find it; There is (at this time) no central source of developer's information available - something which is of importance when doing development for a specialised environment as the Agenda.

When searching for development information for the agenda, the two following sites may be most helpful (in random order):

- Andrej Cedilnik's Agenda pages: `http://www.csee.umbc.edu/~acedil1/agenda/`. COntains useful information for developing for the SVR4 system (see below for more information on what that exactly means)
- The AgendaWiki site at `http://agendawiki.com/`. Contains a detailed description of a SNOW development environment setup.

While the AgendaComputing site also has a developers section, it appears to be out of date, and contains very little actual information on how to start developing for the Agenda. It does give a lot of links to other sites containing information on the agenda.

To make matters worse, the system of the Agenda is still changing: The specification of the binaries has changed. While earlier versions of the Agenda were running a SVR4 system (SVR4 refers to the Application Binary Interface (ABI) used on these Agendas), it seems that more recent Agendas are running a SNOW ABI. The problem with this is that applications compiled for one format cannot be run on the other format. This would not be a problem, if it wasn't that this change does not seem to be officially documented anywhere, and finding out which of the two systems is installed on an Agenda was not possible. Only after installing a new romdisk (a binary image of the system installed on the Agenda, basically a complete Linux installation) with the latest system, it was clear from the release notes that this was a SNOW system.

The SNOW ABI is faster than the SVR4 ABI (one of the main reasons for the switch from SVR4 to SNOW). One of the reasons for this is the fact that shared libraries are loaded at fixed locations in memory. This has as a drawback that the libraries used in the cross-development environment must always exactly match the Agenda's libraries, and that making new libraries is more difficult.

The right steps to produce a development environment were found on AgendaWiki's site. The following steps led to a working development environment:

1. Download and install the latest romdisk (Named snow h2o 1.2.6) from AgendaComputing:

   `http://www.agendacomputing.de/agenda-e/software-e/index-soft-e.htm`

   Instructions on installing this romdisk can be found in the PDF document that comes with the Agenda, and will not be repeated here.

2. Download and install the development environment for this system:

   `http://www.agendacomputing.com/~james/james-snow-compiler-1.2.0.tar.gz`

   This archive contains a complete development system for the agenda. It contains a compiler, linker assembler and other binary utilities. Other than that it contains a version of the libraries that are present on the Agenda; This is especially important in view of the SNOW ABI format. Installing it just a matter of extracting the files in a suitable directory (/opt is recommended on AgendaWiki's site, and will be assumed in the below text).

   The binaries installed and ran without problems on a SuSE 7.3 system.

The compiler binary is called mipsel-linux-gcc and resides in /opt/snow-gcc/bin. In order to use the correct libraries and tools, the -B option should be supplied when calling the compiler: `-B/opt/snow-gcc/lib/snow`. It tells the compiler to look for libraries and tools in the `/opt/snow-gcc/lib/snow` directory.

The following small shell script sets some environment options which prepare the development environment:

```
# Name this script 'startsnow'
export PATH=/opt/snow-gcc/bin:$PATH
export CC="mipsel-linux-gcc -B/opt/snow-gcc/lib/snow/"
export CONFIG_SITE=/opt/snow-gcc/config.site
```

The first line makes sure the compiler directory is in the PATH. The second line defines the CC environment variable, which will be used by `make` when compiling binaries. The last line points to a file that is used by the `configure` tool used by many open source projects. It contains many variables that describe the cross-development environment for the Agenda.

The above must be sourced in the shell before cross-compiling:

```
. ./startsnow
```

Don't execute the file, because the settings will be lost when the shell script exits. After cross compiling, the settings should be undone, since any make or configure will assume settings for cross-compilation.

For people using the C shell, the above would translated to

```
# Name this script 'startsnow'
setenv PATH /opt/snow-gcc/bin:$PATH
setenv CC "mipsel-linux-gcc -B/opt/snow-gcc/lib/snow/"
setenv CONFIG_SITE /opt/snow-gcc/config.site
```

And sourcing it in is done as follows:

```
source ./startsnow
```

After these steps are done, it is possible to cross-compile binaries for the agenda. However, compiling the binary, transferring it to the Agenda and testing it there is not very efficient. That is why some extra steps must be done in order to compile and run the applications on a Linux system; Namely the GUI libraries must be compiled and installed. This is discussed in the following section.

3 Libraries used on the Agenda

The agenda is a Linux system, and as such the standard Linux libraries such as glibc, X11 and others are present. In principle, glibc is sufficient for compiling most command-line applications; For X-Windows development, however, the X11 library is too limited; Commonly, a toolkit library such as GTK or Qt is used to write programs for X. On the Agenda, 3 libraries are used to produce X binaries:

FLTK The Fast Lightweight Toolkit, is a C++ widget toolkit that was derived from the X-forms C library - people familiar with X-forms will feel at home using FLTK. It should work on any Unix system, including linux. It also works on a Windows system, e.g. using the cygwin development environment. It comes with a complete manual, which has a description of all classes and their methods. It is available in HTML and as a printable document (PDF).

Last but not least, FLTK comes with a visual form designer (FLUID) which generates code for all GUI elements. It can be found on

```
http://www.fltk.org/
```

Agenda computing uses version 1.0.9 of the library, and they provide an archive plus patch on their FTP site:

```
ftp://ftp.agendacomputing.com/agenda/dists/h2o/main/source/libs/fltk-1.0.9.orig.
ftp://ftp.agendacomputing.com/agenda/dists/h2o/main/source/libs/fltk-1.0.9-1.dif
```

Flek (Fast Light Environment Kit) An extension library to FLTK; While not strictly required to do development, it contains some useful widgets and other classes not found in FLTK. It can be found on

```
http://flek.sourceforge.net/
```

Flek is also well documented, although all documentation is strictly on-line. Agendacomputing offers the version which they use, together with a patch, on their FTP site:

```
ftp://ftp.agendacomputing.com/agenda/dists/h2o/main/source/libs/flek-20010305.or
ftp://ftp.agendacomputing.com/agenda/dists/h2o/main/source/libs/flek-20010305-1.
```

flpda A collection of widgets and routines developed specially for the Agenda. It is available, together with a patch, on AgendaComputing's site:

```
ftp://ftp.agendacomputing.com/agenda/dists/h2o/main/source/libs/flpda-0.4.orig.t
ftp://ftp.agendacomputing.com/agenda/dists/h2o/main/source/libs/flpda-0.4-1.diff
```

Installing and compiling of these libraries is generally limited to untarring the distribution, running configure and make. flpda can be compiled and installed by running the following:

```
tar xzf flpda-0.4.orig.tar.gz
gzip -d flpda-0.4-1.diff.gz
cd flpda-0.4
./configure
make
make install
```

fltk and flek compile and install in exactly the same manner.

After doing this, a complete testing and cross-compiling environment is set up, and applications can be developed for the Agenda.

4 Using Make

Since a binary compiled for the agenda cannot be run on a linux system and vice versa, two binaries must be compiled, one for the host linux system and one for the Agenda itself.

In order to make these binaries without having to switch the compiler environment (see the variables defined with the 'startsnow' script), a Makefile can be set up to compile binaries for both systems. The below describes how to do this. The scheme presented here is an adaptation of the Makefile that comes with the Thumbpad application for the Agenda.

The Makefile can be set up as follows; First, some make variables are defined:

```
CC=$(PROGPREFIX)g++
STRIP=$(PROGPREFIX)strip
LD=$(PROGPREFIX)ld
OBJECTS=inifile abslist absentee
OBJECTFILES=$(addsuffix $(OEXT),$(OBJECTS))

CFLAGS=-O2 -Wall -I/usr/include $(EXTRA_CFLAGS)
FLTKLIB=-L$(LIBDIR) -lflpda -lflek_ui -lflek_core -lfltk -lXext -lX11

.SUFFIXES: $(OEXT)
```

The `PROGPREFIX`, `OEXT` and `EXTRA_CFLAGS` will be supplied by a recusrive call of make and will depend on the platform one wishes to compile for.

Then come some general targets:

```
all: host agenda

# Build for host system (for Linux)
```

```
host:
make \
TARGET=absentee.host absentee.host \
PROGPREFIX= \
LIBDIR=/usr/X11R6/lib OEXT=.o
# Build MIPS SNOW binary (for Agenda)
agenda:
make \
TARGET=absentee.snow absentee.snow \
PROGPREFIX=mipsel-linux- \
EXTRA_CFLAGS="-Os -B/opt/snow-gcc/lib/snow/ --static" \
LIBDIR=/opt/snow-gcc/lib/snow OEXT=.os
```

The `host` target will recursively call make with the target for the hosts system, and will set some variables. The `agenda` target will call make with the target for the snow binary:

PROGPREFIX is a program prefix for the used binaries; it is empty when compiling for the host system, but equals `mipsel-linux-` when cross-compiling.

OEXT Contains the extension for object files for the target platform; it equals .o for the host system, and `.os` for the snow object files.

EXTRA_CFLAGS some extra flags for the C compiler, they depend on the platform.

Finally come the real rules:

```
$(OBJECTFILES): %$(OEXT): %.cxx
$(CC) $(CFLAGS) -c -o $@ $<

$(TARGET): $(OBJECTFILES)
$(CC) $(CFLAGS) absentee.cxx $(FLTKLIB) -o $(TARGET) inifile$(OEXT) abslist$(OEXT)
$(STRIP) $(TARGET)
$(STRIP) --remove-section .compact_rel $(TARGET)
$(STRIP) --remove-section .note $(TARGET)
$(STRIP) --remove-section .comment $(TARGET)
```

The first rule is a generic rule for producing object files from C++ files, and the second is the final target rule; it links the application and strips all unnecessary information from it.

Since the SNOW format relies on libraries being present at a certain location in memory, a binary that was compiled and linked against one set of libraries may not run on a system that has a set of different libraries (remember that the SNOW libraries must match exactly).

If one wishes to distribute binaries that can be relinked for a particular romdisk, it is possible to make a partially linked binary, which can then be relinked in an environment that has the correct versions of all libraries. In that case, the last Makefile rule must be changed to two rules:

```
$(TARGET).partial.$(OEXT): $(OBJECTFILES)
        ($LD) -r $(OBJS) -o my-app.partial.o

$(TARGET): $(TARGET).partial.$(OEXT)
        $(CC) (TARGET).partial.($OEXT) $(FLTKLIB) -o $(TARGET)
$(STRIP) $(TARGET)
$(STRIP) --remove-section .compact_rel $(TARGET)
$(STRIP) --remove-section .note $(TARGET)
$(STRIP) --remove-section .comment $(TARGET)
```

Now, the partially linked files can be distributed, and someone wishing to install the binary can link against his copy of the SNOW libraries, and be sure that the produced binary will work on his system.

5 The absentee application

The sources for the program to take absentee lists will not be presented her, the interested read can find them on the CD-ROM accompanying this magazine. The code is very straightforward and simple, and is quite similar to code that would have been written in GTK or most other GUI toolkits. (Be warned that there may be some memory leaks, as this is the authors first C++ program).

To keep data, a text file in Windows .ini format is used as it is easy to edit. The support for db files in the flpda library was not used for two reasons:

1. In the author's opinion, the db format doesn't really lend itself to data which have no clear identifying key - which is the case here.
2. More fundamentally, the application used to synchronise files with the Database is written in Delphi, and there is no ready interface for DB files. Windows .ini files on the other hand can be easily managed.

To manage the .ini files on the Agenda, a inifile class is used. The code for the inifile class was taken from the 'cinifile' project on sourgeforge (`http://sourceforge.net/projects/inifile/`), with a bug fix applied by the author, and an extra method added.

The application's main window, running on a Linux host system can be seen in figure 1 on page 8.

When pressing the 'Edit' button, the application's detail window is shown, where the list of pupils can be edited. This window is presented in figure 2 on page 9.

There are some things that should be taken into consideration when developing GUI applications for the Agenda:

- The screen's geometry is 160x220 pixels. This is not very much, as very little will fit on a screen; Therefore an intelligent screen design is important to make the user experience pleasant. The Widget Factory in the `flpda` library creates windows and toolbars that are suitable for the Agenda's screen. Unfortunately, there is no documentation, but the sources are very small, and most code is in the header files.
- The screen does not display colours. Marking things by displaying them in alternate colours therefore will not have the desired effect.
- The mouse pointer is actually the little pencil that comes with the Agenda; it is not as functional as a mouse - there is no right or middle mouse click.
- There is a keyboard, but it is not a PC's keyboard. As such, the working of shortcut keys or special key strokes should be avoided.

These shortcomings can be overcome by intelligent screen design, with a minimum number of widgets and minimum need for data entry.

6 Synchronising files

The Agenda uses the freely available 'rsync' application to synchronise files between a PC and a host computer. While rsync is good at what it does, in an integrated solution, the end

Figure 1: Main screen of the absentee application

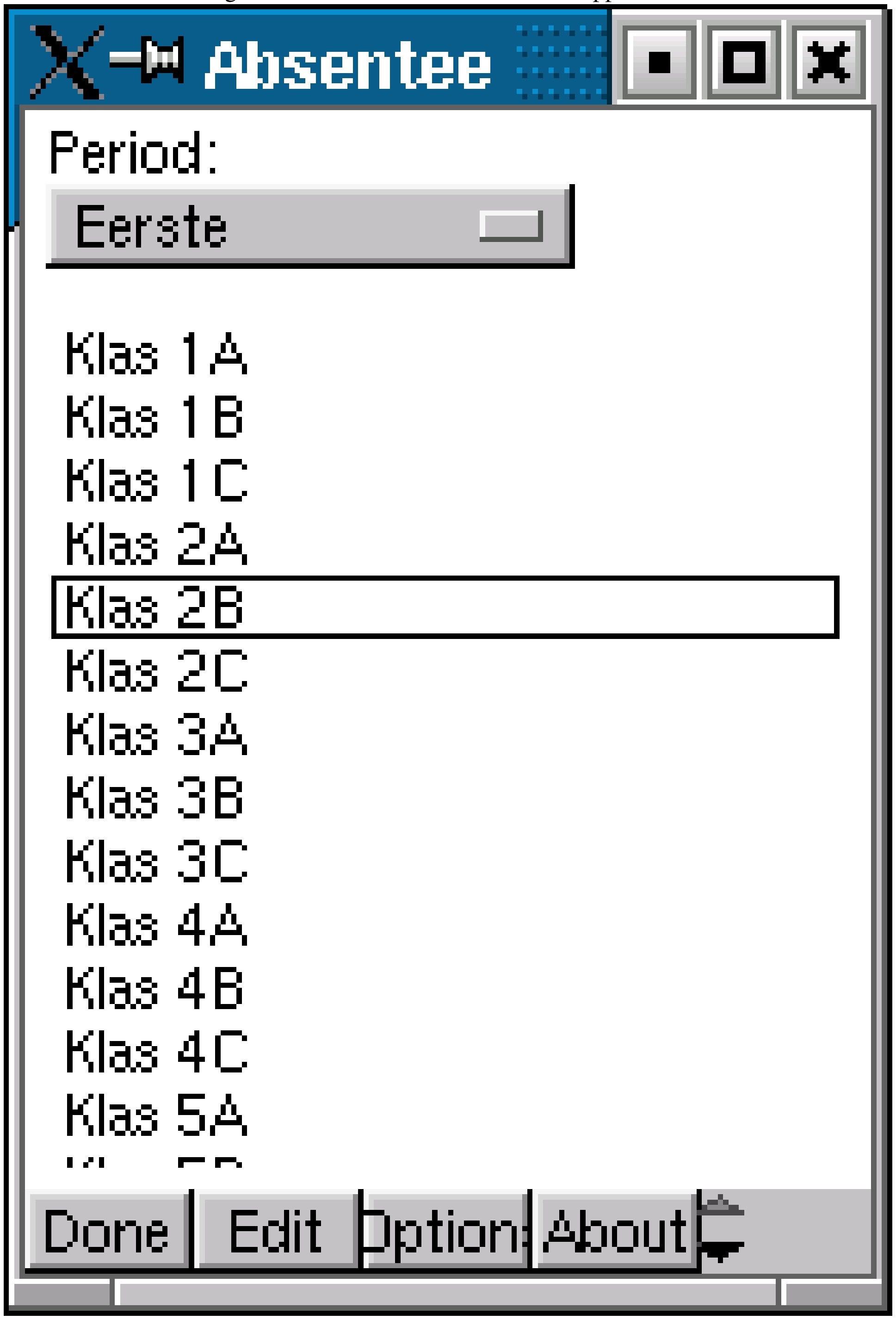
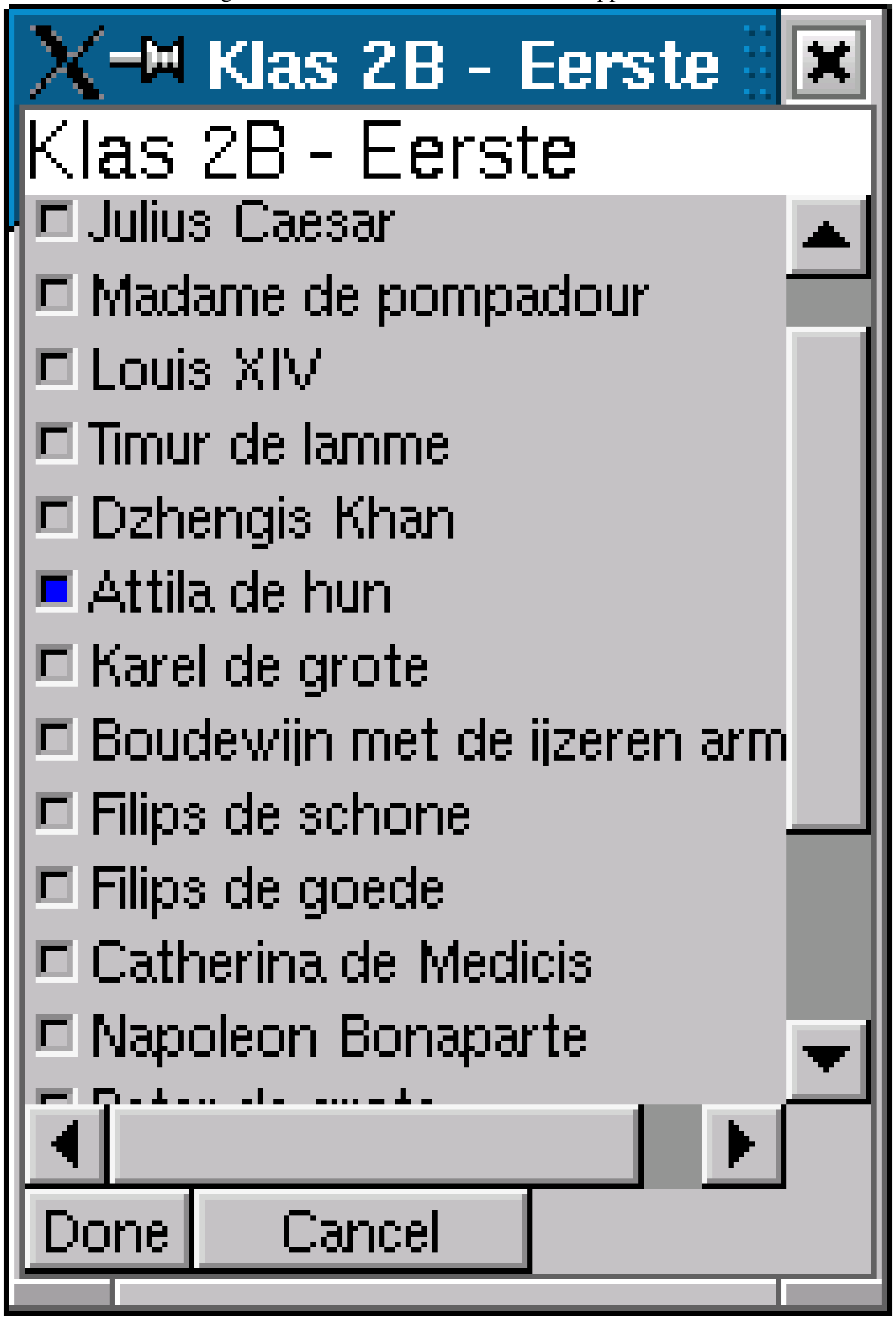


Figure 2: Detail screen of the absentee application



Klas 2B - Eerste

☐ Julius Caesar
☐ Madame de pompadour
☐ Louis XIV
☐ Timur de lamme
☐ Dzhengis Khan
■ Attila de hun
☐ Karel de grote
☐ Boudewijn met de ijzeren arm
☐ Filips de schone
☐ Filips de goede
☐ Catherina de Medicis
☐ Napoleon Bonaparte

user should not be confronted with rsync.

Preferably, the file transfer should be an integrated part of the user's well-known application. In Unix systems, this can be remedied by writing a small graphical shell around rsync using the `popen()` call. On Windows, although it can be done, this kind of technique is not very common. The 'PDA manager' application for Windows uses this technique to copy files between Windows and the Agenda.

Another solution would be to run a minimalist FTP server on the agenda, and copy files using the FTP protocol. For the various Windows development environments, many components exist which manage file transfers using an FTP session. Running the FTP server would consume precious system resources on the Agenda, and since the rsync daemon is needed for other Agenda synchronisation tools, the FTP server will not replace the rsyncd daemon.

Arguably, the most useful and easy solution seems some kind of library which offers an API for file transfer from and to the Agenda; In that case the underlying mechanism is invisible for the programmer and the user:

- It could be a simple API shell around rsync.
- It could take the approach WinCVS uses: Compile the rsync application as a DLL with some well-chosen exported routines, such as callbacks for progress reporting etc.
- Any other protocol for which a server can be run on the Agenda could be used.

7 Conclusion

Developing for the Agenda is definitely not hard - the many programs found on various web sites are a testimony for this. With some searching on the web, information on creating a development environment can be found. While a typical hobbyist or Linux enthusiast will not have any problem with this, for a corporate development team, the lack of a central and authoritative developing information repository and a stable development environment (e.g. the transition from SVR4 to SNOW ABI) could prove a serious argument against the use of the Agenda in the development of an integrated end user software solution. These problems can and, no doubt, will be solved in time.