Outwit: Unix Tool-based Programming Meets the Windows World

Diomidis D. Spinellis
Department of Information and Communication Systems
University of the Aegean
GR-83200 Karlovasi, Greece
dspin@aegean.gr

Abstract

The ubiquity of Windows-based desktop environments has not been matched by a corresponding emergence of tools supporting the Unix tool composition paradigm. *Outwit* is a suite of tools based on the Unix tool design principles allowing the processing of Windows application data with sophisticated data manipulation pipelines. The *outwit* tools offer access to the Windows clipboard, the registry, relational databases, document properties, and shell links. We demonstrate a number of applications of the *outwit* tools used in conjuction with existing Unix commands, and discuss future directions of our work.

> *"GUIs normally make it simple to accomplish simple actions and impossible to accomplish complex actions."* — Doug Gwyn

1 Introduction

The remarkably productive environment based around the Unix tool composition philosophy [KP84] is increasingly made irrelevant by the ubiquity of Windows-based desktop environments. In those environments huge, monolithic, "user-friendly", GUI-based applications, binary file formats and databases, and an API of byzantine complexity render useless a large proportion of the Unix toolbox and associated data manipulation techniques. Often, even the blunt application of a tool like *strings* on a Windows binary file such as a Word document can, as part of a small pipeline, produce results that can not be otherwise obtained from the corresponding GUI-based applications.

The increasing sophistication of Windows-based compilers, libraries, and operating system functionality in conjunction with the availability of open-source Unix tool implementations have led to a number of ports of a large portion of the Unix toolchest to the Windows environment. Although such ports provide a rich set of tools, they often fail to address tool integration with the rest of the environment.

Over the last few years we have nurtured and sharpened *outwit*: a small suite of tools, based on the Unix programming philosophy, yet tightly integrated with the Windows environment. The tools are designed to be seamlessly used with traditional parts of the Unix toolchest as parts of shell-based pipelines. We use these tools on an everyday basis both casually to enhance our system's usability, and as building blocks for more sophisticated applications.

The aim of this paper is to demonstrate that GUI-based platforms, and Windows in particular, are not incompatible with the Unix tool-based approach to prototyping, reuse, shell programming, and implementation based on incremental refinement. In the following sections we will outline the design principles of our approach, describe the tools in our collection, comment on their implementation effort, illustrate some typical uses, and discuss our work in the context of other efforts and future directions.

2 Design Rationale and Principles

Typical GUI-based environments and applications provide a fixed set of features wrapped in a standardized user-friendly interaction paradigm based on windows, icons, menus, and pointing (WIMP). Although this approach enables inexperienced users to quickly master applications and produce results, its limits rapidly become apparent when one tries to perform a task not foreseen by the original developers of the application. In many cases it is difficult to combine different applications together to achieve a complex task, repetitive tasks can not be automated, and interaction sequences can not be manipulated for future reuse. In short, GUI based applications are inflexible factories rather than tools; end-users must accept the end products or wait for central planning to devise a new ones.

Some applications, to overcome these problems, provide an object model and a script-based programming interface. Such interfaces, however, still carry with them the weight of the application, do not provide a clean way to combine different applications, and, in our experience, are typically fragile, and difficult to use in a non-interactive setting.

An alternative approach to productivity in an interactive environment was popularized by the Unix tool-based and shell programming approach [KP84]. Unix tools can be easily combined to form powerful data manipulation sequences. Where such sequences are to be reused or repeated Unix shell-based programming can be used to package and automate them. The advantages of the Unix-based approach to interaction prompted us to examine the possibility of offering these benefits within the context of the ubiquitous Windows environment. A large part of the Unix toolchest has already been ported to Windows as part of open-source and commercial efforts. However, the tools provided typically function isolated from the GUI environment within a text-based command shell. Data interchange with the GUI applications is often limited to the use of textual files which are in any case rare as most GUI applications use undocumented binary file formats. We therefore focused our attention on providing mechanisms for integrating the capabilities provided by the Unix toolchest with the data sources and sinks available in typical GUI applications.

Out tools were designed along the principles used by most Unix-based tools:

- do a single job well,
- avoid decorative headers and trailing information,
- accept input created by other tools,
- generate output that can be used by other tools, and
- be capable of stand-alone execution without user intervention.

It is interesting and instructive to contrast tools based on these principles against the behemoths that populate GUI-based office productivity suites.

We aimed our tools to areas where data amenable to Unix-style pipeline processing could be extracted from difficult-to-access Windows sources. All the tools are console-based native Win32 applications using their standard input and output as the data source and sink. They can be used both from the standard Windows shells (*command* and *cmd32*) and from Unix-derived shells like *bash* and *ksh* [Kor94].

Due to the complexity of the Windows API the external tool simplicity is often inversely proportional to the code required to implement the tool. We found however, that the effort required to implement such tools was often amortized by less mouse clicks and other repetitive GUI operations within a matter of days.

3 Tool Descriptions

The tools we developed aim to make data that is typically accessed through GUI-based applications usable via standard command pipelines. We therefore aimed at providing integration mechanisms for the Windows clipboard, relational databases, the Windows registry, OLE document properties, and shell-namespace links. In the following paragraphs we describe the facilities offered by each tool.

3.1 Clipboard Integration

Most Windows GUI applications and a number of the native system controls (widgets) allow the user to copy and paste data to and from a global clipboard area. This user-driven interprocess communication mechanism is used for sharing data between applications. The clipboard data can be in an application-native format, in which case it can be processed only by the application family that is aware of that format, or it can be in a number of documented public formats. The Windows system will automatically convert between a number of compatible formats such as different types of bitmaps.

Our *winclip* tool provides shell-based clipboard access. When run with a -c (copy) argument it will copy the data it reads from its standard input to the Windows clipboard:

```
ls -l | winclip -c
```

Currently all data is copied to the clipboard as text. When *winclip* is run with a -p (paste) argument it will paste the data of the Windows clipboard to its standard output:

```
winclip -p | wc -w
```

Currently *winclip* supports the following clipboard data formats for its output:

- text,
- bitmap — output in *ppm* [P+93] format, and
- drag file list — files "copied" using the Windows Explorer interface are output as lines of the respective file paths.

3.2 Database Access

The Windows platform is increasingly used as a client for database systems. Database communication is often performed using the Open Database Connectivity (ODBC) application programming interface [Mic97]. ODBC is based on the Call-Level Interface (CLI) specifications from X/Open and ISO/IEC [ISO95] for database APIs and uses the Structured Query Language (SQL) as its database access language. An ODBC driver manager, provided as part of the Windows platform, allows the installation of the drivers supplied by most database vendors. Using an appropriate ODBC driver Windows-based applications can transparently communicate with a wide variety of database engines. Although most database vendors supply text-based SQL database access tools, these are typically database specific and difficult to use for shell-based programming.

Capitalizing on the strengths of the ODBC interface we wrote the *odbc* tool which prints the results of an SQL SELECT command run on any database for which an appropriate data source has been defined:

```
mail `odbc uDB 'select email from users'`
```

Two arguments must always be specified as part of the tool invocation: the name of the data source driver (the database communication configured in the local machine), and a select statement. In addition, one can specify the record and field separator that will delimit the output results, and a user identifier and an authorization string pair that will be used for logging into the database. All data types are currently output as text according to the default driver-supplied data conversions.

3.3 Registry Operations

The Windows registry is a database that stores system management information in a hierarchically structured tree. Each node in the tree, called a key, can contain both subkeys and data entries. A data entry can contain a text string, an integer, or binary data. Windows systems use the registry for storing all configuration data including data related to the operating system, computer hardware, applications, and user preferences. The registry is physically divided into a part containing configuration information about the machine and the operating system, and a part containing user-specific information. The user-specific part is often replaced during the login procedure to allow user roaming or the sharing of a machine among different users. In addition, a special category of keys, called dynamic keys, are updated at run-time by device drivers to monitor quantities such as data transfer rates, processor utilization, and dropped packets. The registry can be manipulated through the GUI registry editor *regedit*. Although this program supports a text import/export format and an, almost undocumented, command-line interface, the textual representation used is not amenable to Unix-tool based processing and the command-line interface is too rigid for many useful applications. Specifically, the text format used by *regedit* places the key names in a separate line followed by a number of key values. In addition, the command-line invocation of the tool executes only in the background and does not allow the text-based representation to be redirected as input or output.

In order to overcome these difficulties we designed and implemented *winreg*, a text-based tool that can be used to read and modify registry data. Each text line represents a single key. Three fields, separated by a user configurable field separator, contain the key name, its type, and its value. The tool can be invoked with the name of a part of the registry as its argument to print the textual representation of the registry tree from that point downwards on its standard output, or it can read the textual representation of some registry keys from its standard input and enter them into the registry. In addition, one can specify whether the output of *winreg* shall include the key names, types, or values. As an example:

```
winreg -nt HKEY_LOCAL_MACHINE\System\
  \CurrentControlSet\Control\ComputerName\
  \ComputerName
```

will display a machine's name.

3.4 Document Properties

A number of applications in the Windows environment expose document meta information such a document's title, author, keywords, and number of pages it contains by using a standard document information property set. This set is part of Microsoft's Component Object Model (COM) structured storage facilities. Three property sets are currently defined:

the summary information containing the document's title, subject, author, keywords, comments, revision, editing time, the time the document was created, printed, and saved, the number of pages, words, and characters, and the name of the creating application,

the document summary information containing the document's category, presentation target, number of paragraphs, lines, notes, slides, and the names of the company and the project's manager, and

the user-defined property set where users can create and store named properties to store additional document information.

Document properties are typically accessed from within applications, or through a file context menu of the Windows GUI shell.

Our *docprop* tool provides Unix shell programmable access to these properties. The program takes as arguments an optional output format specification string and a list of filenames. The format specification string can contain arbitrary text, system or user-defined property names enclosed in braces, and the usual C language backslash escapes. When executed the program will iterate over the filename list and, for each file, will output the format string replacing the property names enclosed in braces with their respective values. In addition, *docprop* provides an internally synthesized property name called `Filename` to allow the printing of each filename. The following example will print the document name and author name of all Word documents in a directory:

```
docprop -f '{Filename}\t{Author}\n' *.doc
```

3.5 Shell Links

Shell links, also known as shortcuts, are data objects that contain information used to access another object in the GUI shell's namespace. They are superficially similar to the Unix symbolic links; their most important difference is that the file or object they point to is *not* resolved automatically by the kernel when an application accesses a link. In addition, if the target object is moved, the system will attempt to locate the target object in its new position when the link needs to be resolved. Links typically operate only at the level of the GUI shell. The types of objects that can be accessed through shell links include files, folders, disk drives, and printers.

Despite our philosophical objections to the shell link concept we found the need for a tool to resolve such links at a textual level. We thus discovered that the operation that is trivially performed on a Unix system by the *readlink(2)* system call needed 60 lines of C code and 11 calls to the Win32 API in the Windows environment. Our resultant *readlink* tool accepts as its single argument the name of a shell link and outputs the name of the link's target. Dynamic resolving of targets on the move works as advertised:

```
$ readlink s.lnk
C:\src\win32lib\port.c

$ move port.c foo
C:\src\win32lib\port.c =>
    C:\src\win32lib\foo\port.c [OK]

$ readlink s.lnk
C:\src\win32lib\foo\port.c
```

4 Implementation Notes

The realization of all tools is based on the possibility to call Win32 API functions from within so-called *console*, i.e. text-based, applications. Although all examples on the use of the Win32 API functions provided with the Microsoft documentation are complete GUI-based applications, in practice most API functions can be called without a problem from text-based programs. This technique forms the basis for integrating the text-based and the GUI worlds. All our programs, implemented in C and C++, call the appropriate Win32 API functions to transfer data to and from the GUI world and use *stdio* I/O functions to interface with the Unix shell world.

At 1440 lines of code the total implementation effort for the *outwit* suite is embarrassingly modest given the capabilities it provides. The code size of each tool is detailed in Table 1. Most tools are at the same time more versatile, powerful, and smaller than the corresponding GUI-based toy examples demonstrating similar capabilities. With increasing use of the *outwit* suite we expect to add more tools and enhance the capabilities of the exist-

Tool name	L.O.C.
readlink	141
odbc	151
docprop	408
winreg	545
winclip	195
Total	1440

Table 1: Tool code size metrics

ing ones. However, given that the power of our approach relies on the synergies of tool composition and not on the features of a single tool, we expect both the number of tools and the size of each tool to remain fairly small.

The greatest hurdle in the development of each tool was the Win32 API. Many API functions are incompletely or inconsistently documented [Spi98] while the sheer size of the API is formidable: it currently comprises 149 different data types, 2193 basic functions, and 1499 error codes. We were also troubled by subtle incompatibilities between operating system versions that could create portability problems, the lack of documentation on the possible function error return values, and the overly complicated interface provided by some functions. We were however pleasantly surprised by the quality of the ODBC interface which stands apart in documentation quality from the rest of the Windows API.

Many API functions are based on special *handles* which must be carefully allocated and deallocated. We believe that this must be a source of reliability problems in the large GUI-based applications. In contrast, the *outwit* tools will simply process data and exit, thereby sidestepping various resource leak problems.

In order to maximize the applicability of the *outwit* tools, all of them are written as native Win32 applications and do not rely on an intermediate Unix porting layer. They can thus be used together with any collection of Unix tools that allows the execution of Win32 programs. All tools can be used under Windows 95, 98, 2000, and NT.

5 Exemplar Uses

We believe that the mode of work enabled and demonstrated by the *outwit* tool suite is more important than the tools themselves. In the following paragraphs we therefore present some — motivating we hope — examples of how Unix tool-based programming can be applied to the Windows GUI world.

5.1 Winreg

Tool-based access to the Windows registry can allow, in combination with other Unix tools, sophisticated registry manipulations that are impossible within the GUI-based registry editor. A commonly used idiom involves processing Windows registry data as the output of *winreg* using Unix tools like *sed* and *awk* and redirecting their output back to the registry via *winreg*. The following example will change all user registry references from `c:/home` to `d:/home`:

```
winreg HKEY_CURRENT_USER |
sed -n 's/C:\\home/D:\\home/gp' |
winreg
```

Winreg is also often used to extract system information from the registry. The name of the currently logged-in user can be stored in a shell variable using the following construct:

```
LOGIN=`winreg \
 HKEY_LOCAL_MACHINE\Network\Logon |
awk '/username/{print $3}' `
```

In addition, network settings can be obtained from the registry keys `HKEY_LOCAL_MACHINE\System\-CurrentControlSet\Services\VxD\MSTCP` and `HKEY_LOCAL_MACHINE\System\CurrentControlSet\-Services\VxD\DHCP`.

One other application of *winreg* involves accessing system performance information similar in nature to the data provided by the Unix *vmstat* command. The following command sequence will save system performance information (CPU load, data I/O, cache and memory system data in the form of raw counter values) to a file for later statistical processing. This functionality is not provided by the system-supplied performance monitors and would otherwise require the implementation of a separate program:

```
while :
do
  winreg -dt HKEY_DYN_DATA\PerfStats\StatData
  sleep 5
done >/var/log/perfdata
```

5.2 Winclip

Winclip can often be used as a quick way to provide functionality lacking from GUI applications. The command `winclip -c </home/signature`, when tied to a

keyboard shortcut, can be used to quickly obtain a copy of the user's signature file contents within any application. Similarly, the following sequence can be used to trivially encrypt or decrypt the text contained in the clipboard using the *rot13* substitution cypher:

```
winclip -p |
tr '[a-z][A-Z]' '[n-z][a-m][N-Z][A-M]' |
winclip -c
```

(This "encryption" method is sometimes used in certain newsgroups to discourage casual reading of joke punch lines or offensive jokes.)

By using the point and click metaphor, GUI interfaces to the filesystem such as the Windows Explorer often provide a swifter method for selecting files than typing file names at the command prompt. *Winclip* allows the integration of the two interaction approaches. The following sequence will open up a console window with its current directory set to the directory copied to the clipboard from the Windows Explorer:

```
\bin\echo -n "start command /k cd " >$$.bat
winclip -p >>$$.bat
start $$.bat
```

The Windows Explorer is also our method of choice for visually selecting groups of files. Having selected a group of files they can then be copied to the clipboard (as drop targets) and processed using text-based commands:

```
sed '/^$/d' `winclip -p` | wc -l
```

Winclip, in conjunction with a pipeline-based image manipulation package such as *netpbm* [P+93], can also be used to automate the processing of graphics images that have been copied to the clipboard. When processing a large number of images this method of work can save the user from the tiresome repetitive operations that would have to be performed on a GUI-based graphics manipulation package. Alternatively, *winclip* can be used to add new facilities to a GUI package by processing the image in the clipboard. As an example, the following sequence of commands will crop, scale, quantize, and convert the clipboard bitmap into a GIF file:

```
winclip -p |
pnmcrop |
pnmscale 0.5 |
ppmquant 256 |
ppmtogif >file.gif
```

5.3 Odbc

SQL and a number of Unix tools offer complementary approaches for obtaining the same result. SQL can be used to select records and fields, group records, join tables, and sort in the same way as the Unix *grep, awk, uniq, join* and *sort* commands. There are however cases where applying Unix tools to data obtained from SQL databases can enhance productivity. One example concerns the combination of different databases (hosted on different servers and applications). Using our *odbc* tool the following sequence selects, merges, sorts, and prints fields from two different databases:

```
(
    odbc ACorpDB "select FullName, Phone from
                  Employees"
    odbc BCorpDB "select surname, name,
                  phonenum from personnel"
) | sort | pr
```

Odbc is also useful when integrating Unix and Windows-based systems. A password file can be extracted from a relational database and copied to a Unix file using the following command:

```
odbc -F: userDB "select * from passwd" \
     >/nfs/host/passwd
```

(The security implications of such functionality are left as an exercise to the reader.)

Finally, given the existence of text-based graph creation tools like GNUplot combining *odbc*, Unix text manipulation tools, and GNUplot can be an efficient way to automate the generation of graphs from databases.

5.4 Readlink and docprop

Readlink and *docprop* are often used together to process GUI file data. *Readlink* is typically used to resolve shell-namespace symbolic links in a sequence similar to the following:

```
case $FNAME in
    *.lnk) FNAME=`readlink $1` ;;
esac
```

Docprop can be used to create and consolidate indices of document titles and authors. *Docprop* can also be used

PROGRAMME

Form INF3

WORKPACKAGE DESCRIPTION

For WORKPACKAGE N° 0

WP title:	Project Management		
WP leader:	P0		
WP contributors:	All		
Start month:	1	End month:	24
Estimated labour effort in person-months: 12			

Activities related to this workpackage will address the following items:
[...]

Constituent task(s):		Task leader:
T0.1	Definition of technical & operational internal procedures and Quality aspects of the Project	P0
T0.2	Contractual Reporting	P0
T0.3	Organisation of Kick-off, Interim, Concertation, and Final meetings	Hosting partner

Deliverables (max 3 per WP)		Month Due:
D0.1	Project Quality Plan	1
D0.#	Progress reports and cost statements	6,12,18,24
D0.6	Final report	24

Figure 1: Workpackage description form.

to gather statistical data from document files. The following sequence will print a list of application names that were used to edit the most recently used files, sorted in order of application popularity.

```
for f in /windows/recent/*.lnk
do
    readlink $f
done |
xargs docprop -f '{Application}\n' |
sort |
uniq -c |
sort -n
```

5.5 Document Processing

The integration of the GUI and Unix tool worlds can be a particularly productive option when processing textual data using (or having to use) a GUI-based word processor. In the following paragraphs we will present an exemplar case concerning processing textual data for a funding proposal. The work to be funded was divided into eight workpackages; each workpackage had to be described using a form similar to the one appearing in Figure 1.

In addition to a description for each workpackage, the funding body required the provision of a list of deliverables ordered by delivery month, and a GANTT chart for the whole project. We also wanted a current list of tasks in order to distribute work between the project participants. All these items are different representations of the data included in the workpackage descriptions. The combined use of *winclip* and some Unix tools allowed us

PROGRAMME

Form INF4

DELIVERABLE DESCRIPTION

Due end of month n°	WP n°	Deliverable Title	Deliverable Type and Status
	0	D0.1 Project Quality Plan	Report, Restricted
,12,18,24	0	D0.# Progress reports and cost statements	Report, Confidential
4	0	D0.6 Final report	Report, Restricted
	1	D1.1 Requirement Specification	Report, Confidential
	1	D1.2 Architecture Design	Report, Confidential
	2	D2.1 Draft Specification of interlingua	Report, Confidential
	2	D2.2 Greek and English compatible revised interlingua	Report, Confidential
2	2	D2.3 Interlingua for implementation	Report, Confidential
	3	D3.1 Metalevel Specification	Report, Confidential
2	3	D3.2 Interlingua for implementation	Report, Confidential
	4	D4.1 Report on selection of resources and their use in the system	Report, Restricted
4	4	D4.2 Prototype Resources	Data, Confidential

Figure 2: Project deliverable list based on *awk* data.

to automate the creation of these items. All sequences we used involved the massaging of the clipboard data (which contained the workpackage descriptions) using *awk* into a format appropriate to be pasted back into another table or application.

The following script was used to create a list of tasks:

```
winclip -p |
awk "-F\t" '/^T[0-9]/{print $1 " " $2}' |
winclip -c
```

The script below was used to create a list of deliverables to be pasted back into a summary form in the original application:

```
winclip -p |
awk "-F\t" '/^D[0-9]/ {
    print $3 "\t" substr($1, 2, 1) \
          "\t" $1 "\t" $2
}' |
winclip -c
```

The corresponding table is depicted in Figure 2.

Finally, the following script was used to create a workpackage list together with start dates and durations in a format suitable for pasting into Microsoft Project:

```
winclip -p |  awk '
BEGIN       {FS = "\t" }
/For WORK/  {split($0, a, "\t")}
```

```
/^WP title/ {WP = $2}
/^Start month/ {
    print "WP" a[4] "\t" \
          WP "\t" \
          ($4 - $2 + 1) * 31 "ed\t" \
          "1/" ($2 - 1) % 12 + 1 "/" 2000 +
               int(($2 - 1)/12)
}' |
winclip -c
```

The output was of the following form:

```
WP0 Project Management 744ed 1/1/2000
WP1 Requirements Analysis 124ed 1/1/2000
WP3 Metalevel Specification 279ed 1/4/2000
...
```

The resulting GANTT chart is depicted in Figure 3.

6 Related Work

A number of efforts have been undertaken to provide the functionality of the Unix tools in the Windows environment. It is interesting that only one of them is based on the POSIX subsystem provided under Windows NT as an alternative to Win32. The main drawback of the POSIX subsystem is that processes running in it are essentially isolated from the rest of the system. Most porting efforts complement our *outwit* suite by providing the necessary tools needed to utilize our offerings. In some cases facilities for cooperating with GUI applications are also provided.

Our *wux* port of Unix tools to the 16 bit Windows environment [Spi94] demonstrated the possibility of implementing true multitasked processing of pipeline command sequences under the Windows environment, but did not offer any additional integration facilities.

The UWIN port [Kor97] of the Unix tools and libraries supports all X/Open Release 4 headers, interfaces, and commands. It supports the shell namespace links (shortcuts) by mapping them internally to Unix symbolic links. It is interesting to note that UWIN resolves links by reverse engineering the OLE link file format rather than calling the corresponding Win32 API functions. UWIN provides access to the clipboard via the `/dev/clipboard` device and to the Windows registry via a virtual file system that is mounted at `/reg`. Both facilities are however only available to programs that have been compiled under the UWIN environment. As a result, native operating system console commands (e.g. *dir*) and commands that have been compiled without using the UWIN libraries can not directly access the clipboard and the registry. On the contrary, *outwit* tools provide clipboard and registry access to all character-based console programs, including 16-bit legacy applications.

Cygwin [Noe98] is a full Win32 porting layer for Unix applications. It supports the GNU development tools and allows the effortless port of many Unix programs by supporting almost all POSIX.1/90 calls and other Unix version-specific functionality. A novel integration aspect of *cygwin* is the provision of the `/dev/windows` pseudo-device which can be used as a source for Windows messages (user input from the keyboard, the mouse, and IPC events).

Finally, *OpenNt* [Wal97] is a complete porting and runtime environment that can be used to migrate application source, developed on traditional Unix systems, directly to Windows NT. Ported software includes many X11R5 clients and over 200 other utilities. It is implemented using an enhanced Windows NT POSIX subsystem. As the POSIX subsystem is isolated from the Win32 subsystem, integration between the two worlds is offered through the filesystem, the desktop, a special function to execute Win32 applications, and socket-based IPC.

A number of technologies support high-level programming in the Windows environment. These include languages such as Visual Basic [Boc99], Perl [WCSP96], and TCL/TK [Ous94], and integration mechanisms such as OLE automation and Windows scripting. These approaches, based on a programming language, are superior for programming in the large and developing applications in a top down manner. Tool-based approaches such as the one we advocate complement such an environment offering a different development path. As tools are directly used from the shell command line in a casual user interaction pattern repetitive tasks are gradually automated as shell scripts and subsequently, as they mature, packaged as applications. We believe that this bottom-up evolutionary style of development should be available together with other approaches.

7 Conclusions and Further Work

The tools we described can be extended in breadth and depth. We are currently planning to enhance *winclip* with additional data types to be able to handle sound data as input and output, as well as bitmap data as input. *Odbc* can also be improved by adding the ability to modify database data using the SQL *update* command. Such a facility will be useful for transferring data between databases through a pipeline with two *odbc* commands at its ends. In addition, with the advent of Windows 2000 which supports the *IFilter* interface, *docprop*

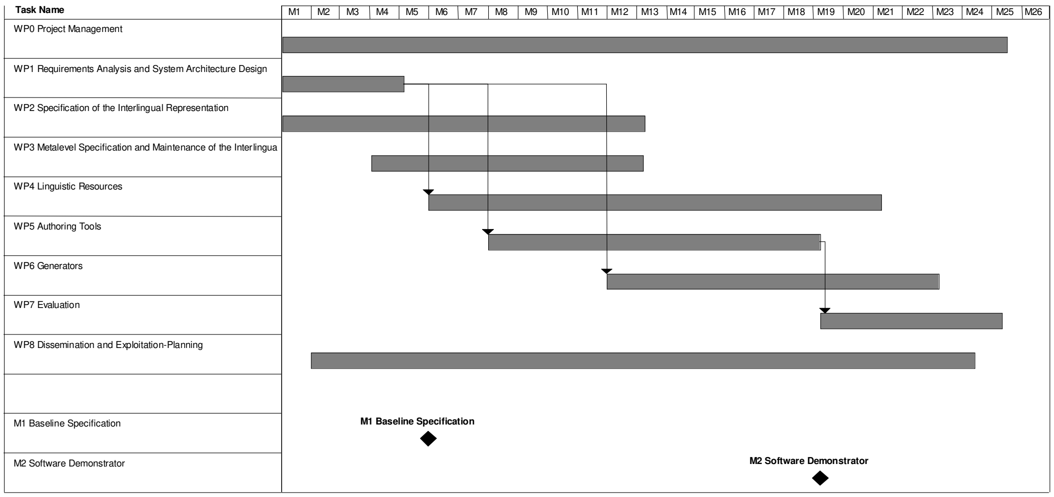

Figure 3: Project GANTT chart based on *awk* data.

can be extended to retrieve a textual representation of any object that supports the respective interface. Finally, the migration of many operating system databases to the Windows 2000 Active Directory has created the need for a new text-based tool to manipulate them.

Many Windows applications expose an object model of the data and operations they support. Environments such as Visual Basic and language extensions such as the Perl OLE module allow programmers to access and manipulate Windows applications and their data from external programs. Although we feel that shell-based programming can not compete against environments tailored to support this mode of programming, a simple text-based command to access OLE automation objects could prove a valuable addition to our *outwit* suite. On the same front, but from a different angle, we are examining how Unix tools can be repackaged as OLE components to be used within visual programming environments [Spi99].

One final dimension of work that will increase the applicability of *outwit* concerns internationalization. All Win32 API functions dealing with strings are provided in 8-bit character and Unicode (wide character) versions. Although low-end Windows systems such as Windows 95/98 are not supporting Unicode for most of the API functions, Windows NT and Windows 2000 provide full Unicode support. As a result, the clipboard, filenames, registry keys, and databases may contain the full repertoire of Unicode characters. Currently *outwit* is coded and compiled as an 8-bit character application. Following the lead established by the Plan 9 internationalization efforts [PT93] a reasonable approach would be to adapt all *outwit* tools to use a multibyte (e.g. UTF-8) representation for their textual input and output. This approach would work with many existing Unix tools; we expect programs that explicitly deal with characters and will need to be modified (e.g. *grep*, *sort*, *sed*, and *tr*) to follow a similar approach.

In the previous paragraphs we demonstrated that the rise of GUI-based environments does not mean that tool building and Unix shell programming are less relevant today than they were 25 years ago. The lack of support for tool-based programming in the Windows environment is a result of market dynamics rather than an inherent limitation of the environment. Integrated GUI-based systems are probably the only applications that can be marketed and sold at a profit. However, modest effort invested in tool building will produce tools that can be used as building blocks in concert with the large number of existing Unix tools and the powerful shells.

Availability

The tools described are available online at:

`http://softlab.icsd.aegean.gr/~dspin/sw/outwit`

References

[Boc99] David Boctor. *Microsoft Office 2000 Visual Basic Fundamentals*. Microsoft Press, Redmond, WA, USA, 1999.

[ISO95] International Organization for Standardization, Geneva, Switzerland. *Information technology — Database languages — SQL — Part 3: Call-Level Interface (SQL/CLI)*, 1995. ISO/IEC 9075-3:1995.

[Kor94] David G. Korn. Ksh - an extensible high level language. In *Very High Level Languages Symposium (VHLL)*, pages 129–146, Santa Fe, NM, USA, October 1994. Usenix Association.

[Kor97] David G. Korn. Porting Unix to Windows NT. In *Proceedings of the USENIX 1997 Annual Technical Conference*, Anaheim, CA, USA, January 1997. Usenix Association.

[KP84] Brian W. Kernighan and Rob Pike. *The UNIX Programming Environment*. Prentice-Hall, 1984.

[Mic97] Microsoft Corporation. *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide*. Microsoft Press, Redmond, WA, USA, 1997.

[Noe98] Geoffrey J. Noer. Cygwin32: A free Win32 porting layer for UNIX applications. In *Proceedings of the 2nd USENIX Windows NT Symposium*, Seattle, WA, USA, August 1998. Usenix Association.

[Ous94] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.

[P+93] Jef Poskanzer et al. NETPBM: Extended portable bitmap toolkit. Available online ftp://ftp.x.org/contrib/utilities/, December 1993. Release 7.

[PT93] Rob Pike and Ken Thompson. Hello world. In *USENIX Technical Conference Proceedings*, pages 43–50, San Diego, CA, USA, Winter 1993. Usenix Association.

[Spi94] Diomidis Spinellis. Wux: Unix tools under Windows. In *USENIX Conference Proceedings*, pages 325–336, San Francisco, CA, USA, Winter 1994. Usenix Association.

[Spi98] Diomidis Spinellis. A critique of the Windows application programming interface. *Computer Standards & Interfaces*, 20:1–8, November 1998.

[Spi99] Diomidis Spinellis. Explore, excogitate, exploit: Component mining. *IEEE Computer*, 32(9):114–116, September 1999.

[Wal97] Stephen R. Walli. OPENNT: UNIX application portability to Windows NT via an alternative environment subsystem. In *Proceedings of the USENIX Windows NT Symposium*, Seattle, WA, USA, August 1997. Usenix Association.

[WCSP96] Larry Wall, Tom Christiansen, Randal L. Schwartz, and Stephen Potter. *Programming Perl*. O'Reilly and Associates, Sebastopol, CA, USA, second edition, 1996.