How I hacked my way into academia

Diomidis Spinellis
Professor, Department of Management Science and Technology
http://www.dmst.aueb.gr/dds
@CoolSWEng
Hacker [originally, someone who makes furniture with an axe] n.
1. A person who enjoys learning the details of programming systems and how to stretch their capabilities, as opposed to most users who prefer to learn only the minimum necessary.
2. One who programs enthusiastically, or who enjoys programming rather than just theorizing about programming.
3. A person capable of appreciating hack value (q.v.).
4. A person who is good at programming quickly. Not everything a hacker produces is a hack.
5. An expert at a particular program, or one who frequently does work using it or on it; example: "A SAIL hacker". (Definitions 1 to 5 are correlated, and people who fit them congregate.)
6. A malicious or inquisitive meddler who tries to discover information by poking around. Hence "password hacker", "network hacker".

— Guy Steele et al The Hacker’s Dictionary, 1988
Challenges are the rule
HF 65, anvendelse:

HF 65 kan anvendes af radiomaterører til mikrofonsender på 144 M Hz. Det er nødvendigt at have senderlicens for at anvende senderen rigtigt.

HF 65 kan anvendes af sømand, samt sejlsportsfolk i rum sø uden licens.

HF 65 er forsynet med en følsom forforstærker og en kraftig udgang. Istedet for en almindelig mikrofon kan man anvende en stereotелефon, der kun koster få kroner i døbelhandel.

HF 65 kan også anvendes af radioanalekterne som målssender. Den skal så bygges ind i en tet metallkasse, som hindrer udsat senderudstråling.

Istedet for at tilslette en mikrofon til HF 65, kan man tilslette en toekskillator, og vil kunne få lidt mere eller mindre strøget på TV.

Når senderen ikke bygges ind i en metallkasse, vil den uden antenne række flere hundre meter, og derved forstyrre FM eller TV båndene.

Med en stor batteri impedans vil senderen kunne række mere end 10 Km. Udgangstransistoren kan tilde en tabseffekt på max. 5 Watt.

**Tekniske data:**
- **Udgangseffekt:** Max. 1 Watt ved 45 Volt batteri impedans
- **Udgangseffekt:** Max. 0,1 Watt ved 9 volt batteri impedans
- **Frekvensområde:** Fra 60 M Hz til Ca. 145 M Hz
- **Spænding:** 4,5 volt til 45 Volt
- **Strømforsyning:** 10 mA til 50 mA max.
- **Indgangspleomhed:** Mikrofon, dynamisk, 10 mV
- **Indgangspleomhed:** Max. 22 K 0

HF 65 bør kun afgive 1 Watt over en kortere periode uden kæleplade.
70cm transverter για εκσυγχρονισμό μέσω δορυφόρου

Το μίνι αρμόνιο υπολογιστής Σόλθφαουν αυτοχρήσιμο αυτοματισμού

Artiest

ισωτικός σχεδιασμός

εναλλαγής των κωνοειδών

disco drum

Ionostiko atmofoikiko aer

Ekterminal

σαλομπατο ερφ

αν μέσως δικτη μεσοδι

ένα μεγάλο VU

-ενιαίωση των καθοδιαλογητών παίζοντας στην TV
-ενιαίωση τηλεφωνικών
-διαμορφωτικό VHF/UHF

δυναμική ελαττωση τορόδου DNR

ενιαίωση κεραίων

φαντάρι LED

θυσία κεραίων

ενιαίωση 100W

-το ΟΤΑ 13600

-δέκτης υπερανάδεικτης 87 - 180MHz
ΤΕΧΝΙΚΗ ΕΚΛΟΓΗ
ΕΓΚΥΚΛΟΠΑΙΔΙΚΗ ΤΕΧΝΙΚΗ ΕΠΙΘΕΩΡΗΣΗ
ΤΕΥΧΟΣ 159
ΜΑΡΤΙΟΣ ΔΡΧ. 35

BASIC η απλή γλώσσα των υπολογιστών

ΣΥΣΤΗΜΑ ΣΥΝΑΓΕΡΜΟΥ ΓΙΑ ΥΠΕΡΑΣΦΑΛΕΙΑ

προενισχυτής γιά μαγνητική κεφαλή πικάπ

ΓΕΦΥΡΑ ΜΕΤΡΗΣΕΩΣ ΣΤΑΣΙΜΩΝ
Το BASIC είναι η απλή γλώσσα των υπολογιστών. Ωστόσο, η γλώσσα φράσης Κρατάεται ορισμένες εννοιών στα οποία ο εκτυπωτής επικοινωνεί με τον υπολογιστή. Για να μπορεί να κατανοήσει τις διαφορές, ο εκτυπωτής θα πρέπει να υπάρχει κάποιος συμβάσεις για την εκτέλεση των διαφορές. Για αυτή την άμεση προσέγγιση, το BASIC επιτρέπει την εκτέλεση διάφορων γλώσσων. Μεταξύ αυτών, την αρχική γλώσσα του βασικού, την ελληνική γλώσσα, την αγγλική γλώσσα και την γερμανική γλώσσα. Με τον χρήσιμο και ευχάριστο τρόπο, ο εκτυπωτής μπορεί να κατανοήσει τις διαφορές μεταξύ των γλωσσών και να εκτελέσει τις διαφορές με τον υπολογιστή.
Expand your horizons
>REM THIS IS A GAME PROGRAM
>0 PRINT "IF YOU WANT TO START THE GAME"
>20 PRINT "PRESS KEY MARKED 1"
>30 INPUT A
>40 IF A = 1 THEN 60 GOTO 60
>50 PRINT "SORRY YOU DON'T WANT TO BE STARTED THATS"
>60 LET D = RND*10+1
>70 LET E = INT D
>80 LET D = RND*10+1
>90 LET E = INT D
>100 PRINT "PLEASE ENTER SHOT SPOT"
>110 INPUT F, G
>120 IF F = E THEN 130
>130 IF G = E THEN 130
>140 GOTO 150 PRINT "I'M SORRY"
>150 GOTO 60
>160 IF (F = 2.0 / 6) = 1 THEN 170
>170 PRINT "YOU HAVE SHOT THE SMALL SPOT"
>180 GOTO 100
>190 PRINT "YOU WINNED"
>200 END
10 REM examine evaluator E
20 FOR I = 1 TO 4
30 IF CHAR(123) = CHR$(123) THEN 30
40 IF CHAR(123) = CHR$(123) THEN 40
50 GOTO 70
60 CHAR(123) = CHAR(123)
70 IF CHAR(123) = CHAR(123) THEN 110
80 PRS
90 APP
100 S$ = CHR$(123)
110 IF $S < $C THEN 200
120 IF ASC($S) > $D THEN 170
130 $S = SEG($S)
140 GOTO 70
150 IF $S < $C THEN 200
160 IF ASCII($S) > $D THEN 170
170 $S = SEG($S)
180 GOTO 70
190 IF CHAR($S) = CHAR($S) THEN 200! $S = 7
200 $S = "$S"
210 NEXT I
220 IF $S = "$S" THEN 290
230 P$ = "$S" & CHAR(ASC($S))
240 $S = SEG($S)
250 GOTO 20
260 ...
Programs can process and generate other programs
A framework for the static verification of API calls

Dionisios Spiniellis *, Panagiotis Louridas

Department of Management Science and Technology, Athens University of Economics and Business, Panepistimiou, 76, GR-104 34 Athens, Greece

Received 20 February 2006; revised in revised form 18 September 2006; accepted 30 September 2006
Available online 13 November 2006

Abstract

A number of tools can statically check program code to identify commonly encountered bug patterns. At the same time, programs are increasingly relying on external APIs for performing the bulk of their work: the bug-prone program logic is being fleshed-out, and many errors involve tricky subroutine calls to the constantly growing set of external libraries. Extending the static analysis tools to cover the available APIs is an approach that replicates scarce human effort across different tools and does not scale. Instead, we propose moving the static API call-verification code into the core implementation, and distributing the verification code together with the library proper. We have designed a framework for providing static verification code together with Java classes, and have extended the FindBugs static analysis tool to check the corresponding method invocations. To validate our approach we wrote verification tests for 100 different methods, and ran FindBugs on 6.9 million method invocations on what amounts to about 13 million lines of production-quality code. In the set of 55 thousand method invocations that could potentially be statically verified our approach identified 800 probable errors.

© 2006 Elsevier Inc. All rights reserved.

Keywords: Static analysis; API; Library; Programming by contract; FindBugs

1. Introduction

Automatic program verification tools have had a significant impact on software development, and are more and more used in practice to eliminate many errors that in the past would have caused program crashes, security vulnerabilities, and program instabilities (Johnson, 1977; Bash et al., 2000; Ball and Rajamani, 2002; Das et al., 2002; Cailliet and Smirnoff, 2005; Cok and Kimy, 2005; Burnett et al., 2006). However, two software development trends are now hindering the applicability of automated program verification tools:

1) the increasing use of binary-packaged components (for the most part libraries) through their application programming interface (API), and

2) the increasing sophistication, and in particular the embedding of many different domain-specific languages (DSLs) as strings in the program code.

Both trends reduce the efficiency of the current approaches. The use of feature-rich libraries in their binary form handicaps verification programs that require access to source code, such as static analysis (Flanagan et al., 2002), and also programs that contain a fixed-set of specific bug patterns, like xss (Viega et al., 2001). Furthermore, the diversity of the libraries handicaps any tool that depends on a centralized repository of verification patterns. In addition, the embedding of DSLs, like xss and xpath, in strings appearing in the program's source code can introduce bugs that are beyond the reach of the current breed of tools based on approaches like theorem proving (Flanagan et al., 2002), dataflow analysis (Jackson, 1995), and finite state machines (Ball and Rajamani, 2002). To overcome these difficulties we propose a framework for incorporating API call verification code within each library containing the corresponding API implementation. Through the use of reflection techniques program checkers can invoke this...
Title: TREASURE-HUNTING GAME

[Formula]

This is a game in which the location of treasure is decided by random numbers and a player hunts out the treasure.

Your initial position is at (0, 0). Input a distance you want to cover in each of the x and y directions. If you get out of the matrix shown on the right, the error indication appears, so try again.

A distance between you and the treasure is indicated as a hint by the value of \( \text{ABS}(x-a) + \text{ABS}(y-b) \).

You are allowed an energy of 60 at the start of treasure hunt. The energy decreases by a sum of distances in the x and y directions which you make. And when you locate the treasure, "HIT" is displayed and the energy increases by 5. Every time you input, the following indication appears:

Example:

\[ \begin{array}{ccc}
4 & 5 & 4 \\
\text{x coordinate of present position} & \text{y coordinate of present position} & \text{Distance between you and treasure (Hint: energy of remaining energy)}
\end{array} \]

Every time the treasure is hunted out, a new location of the treasure is fixed by random numbers. You continue to hunt the treasure until the energy you have exhausts. The number of hunted treasures is displayed at the end of the game.

[Operation]

CLOAD "X11" [END]

Title: TREASURE-HUNTING GAME

Memory content:

<table>
<thead>
<tr>
<th>A</th>
<th>1</th>
<th>x coordinate of target</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>2</td>
<td>y coordinate of target</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>Distance</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>No. of targets found</td>
</tr>
<tr>
<td>E</td>
<td>5</td>
<td>( \geq 0 )</td>
</tr>
<tr>
<td>F</td>
<td>6</td>
<td>( \geq 0 )</td>
</tr>
<tr>
<td>G</td>
<td>7</td>
<td>( \geq 0 )</td>
</tr>
<tr>
<td>H</td>
<td>8</td>
<td>( \geq 0 )</td>
</tr>
<tr>
<td>I</td>
<td>9</td>
<td>( \geq 0 )</td>
</tr>
<tr>
<td>J</td>
<td>10</td>
<td>( \geq 0 )</td>
</tr>
<tr>
<td>K</td>
<td>11</td>
<td>( \geq 0 )</td>
</tr>
<tr>
<td>L</td>
<td>12</td>
<td>( \geq 0 )</td>
</tr>
<tr>
<td>M</td>
<td>13</td>
<td>( \geq 0 )</td>
</tr>
<tr>
<td>N</td>
<td>14</td>
<td>( \geq 0 )</td>
</tr>
<tr>
<td>O</td>
<td>15</td>
<td>( \geq 0 )</td>
</tr>
<tr>
<td>P</td>
<td>16</td>
<td>( \geq 0 )</td>
</tr>
<tr>
<td>Q</td>
<td>17</td>
<td>( \geq 0 )</td>
</tr>
<tr>
<td>R</td>
<td>18</td>
<td>( \geq 0 )</td>
</tr>
<tr>
<td>S</td>
<td>19</td>
<td>( \geq 0 )</td>
</tr>
<tr>
<td>T</td>
<td>20</td>
<td>( \geq 0 )</td>
</tr>
<tr>
<td>U</td>
<td>21</td>
<td>( \geq 0 )</td>
</tr>
<tr>
<td>V</td>
<td>22</td>
<td>( \geq 0 )</td>
</tr>
<tr>
<td>W</td>
<td>23</td>
<td>( \geq 0 )</td>
</tr>
<tr>
<td>X</td>
<td>24</td>
<td>( \geq 0 )</td>
</tr>
<tr>
<td>Y</td>
<td>25</td>
<td>( \geq 0 )</td>
</tr>
<tr>
<td>Z</td>
<td>26</td>
<td>Energy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
<th>Display</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \text{B1} = )</td>
<td>INITIAL</td>
</tr>
<tr>
<td>2</td>
<td>23456789</td>
<td>( 0 )</td>
</tr>
<tr>
<td>3</td>
<td>( \text{dx} )</td>
<td>( \text{dy} )</td>
</tr>
<tr>
<td>4</td>
<td>( \text{gx} )</td>
<td>( \text{gy} )</td>
</tr>
<tr>
<td>5</td>
<td>( \text{gx} )</td>
<td>( \text{gy} )</td>
</tr>
<tr>
<td>6</td>
<td>( \text{dx} )</td>
<td>( \text{dy} )</td>
</tr>
<tr>
<td>7</td>
<td>( \text{gx} )</td>
<td>( \text{gy} )</td>
</tr>
<tr>
<td>8</td>
<td>( \text{dx} )</td>
<td>( \text{dy} )</td>
</tr>
<tr>
<td>9</td>
<td>( \text{gx} )</td>
<td>( \text{gy} )</td>
</tr>
<tr>
<td>10</td>
<td>( \text{gx} )</td>
<td>( \text{gy} )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
<th>Display</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>( \text{HIT} )</td>
<td>( \geq 0 )</td>
</tr>
<tr>
<td>12</td>
<td>( \text{dx} )</td>
<td>( \text{dy} )</td>
</tr>
<tr>
<td>13</td>
<td>( \text{gx} )</td>
<td>( \text{gy} )</td>
</tr>
<tr>
<td>14</td>
<td>( \text{gx} )</td>
<td>( \text{gy} )</td>
</tr>
<tr>
<td>15</td>
<td>( \text{gx} )</td>
<td>( \text{gy} )</td>
</tr>
<tr>
<td>16</td>
<td>( \text{gx} )</td>
<td>( \text{gy} )</td>
</tr>
<tr>
<td>17</td>
<td>( \text{gx} )</td>
<td>( \text{gy} )</td>
</tr>
<tr>
<td>18</td>
<td>( \text{gx} )</td>
<td>( \text{gy} )</td>
</tr>
<tr>
<td>19</td>
<td>( \text{gx} )</td>
<td>( \text{gy} )</td>
</tr>
<tr>
<td>20</td>
<td>( \text{gx} )</td>
<td>( \text{gy} )</td>
</tr>
</tbody>
</table>

Program:

10: \( \text{"A"} = \text{Z} \) | \( \text{dx} = \text{dx} \) |
20: \( \text{dx} = \text{dx} \) |
30: \( \text{GOSUB} \) | \( \text{S} \) |
40: \( \text{dx} = \text{dx} \) |
50: \( \text{GOSUB} \) | \( \text{S} \) |
60: \( \text{dx} = \text{dx} \) |
70: \( \text{dx} = \text{dx} \) |
80: \( \text{dx} = \text{dx} \) |
90: \( \text{dx} = \text{dx} \) |
100: \( \text{dx} = \text{dx} \) |
110: \( \text{dx} = \text{dx} \) |
120: \( \text{dx} = \text{dx} \) |
130: \( \text{dx} = \text{dx} \) |
140: \( \text{dx} = \text{dx} \) |
150: \( \text{dx} = \text{dx} \) |
160: \( \text{dx} = \text{dx} \) |
170: \( \text{dx} = \text{dx} \) |
180: \( \text{dx} = \text{dx} \) |
190: \( \text{dx} = \text{dx} \) |
200: \( \text{dx} = \text{dx} \) |
210: \( \text{dx} = \text{dx} \) |
220: \( \text{dx} = \text{dx} \) |
230: \( \text{dx} = \text{dx} \) |
240: \( \text{dx} = \text{dx} \) |
250: \( \text{dx} = \text{dx} \) |
260: \( \text{dx} = \text{dx} \) |
CODE Reading
Volume 1
The Open Source Perspective
Diomidis Spinellis
Learn by reading code
(and studying systems)
that other people have written
ΚΟΚΚΙΝΟΙ ΠΙΛΑΤΕΣ
ΑΣΠΡΟΙ ΝΑΝΟΙ

ΟΜΑΔΑ 1

ΠΟΙΑ ΕΙΝΑΙ Η ΜΕΓΑΛΥΤΕΡΗ Α
ΠΟΣΤΑΣΗ ΑΠΟ ΤΗΝ ΓΗ ΤΟΥ ΠΛΟΥΤΟΥΑ

-----------
ΚΑΤΗΓΟΡΙΕΣ
1. ΔΙΑΣΤΗΜΑ
2. ΠΗ
3. ΦΥΣΙΟΓΝΩΣΙΑ
4. ΤΕΧΝΟΛΟΓΙΑ
5. ΔΙΑΤΟΜΗ

ΕΠΙΛΟΓΗ
730 SUB LOC((ES),(H))
770 DATA ASTRONOMIA, BIOLOGIA, GEVGRAFIA, FYSIKH, XHMEIA
780 CALL POSITION(#26, Y,X)
790 R=INT((Y+4)/8)+1:: C=INT((X+4)/8)+1
800 IF R=15 OR R<7 OR C>23 OR C<15 THEN H=0:: SUBEXIT
810 IF C=2=INT(C/2) OR R=2=INT(R/2) THEN H=0:: SUBEXIT
820 CALL MOTION(#28, 0, 0)
830 CALL SOUND(100, 1000, 0)
840 CALL DELSPRITE(#26)
850 CALL SPRITE(#28, 42, 7, R'8-12, C'1-8')
860 CALL MAGNIFY(2)
862 IF P=1 THEN 670
863 RESTORE 770
865 FOR I=1 TO 5:: READ PS(I)::* NEXT I::* P=1
870 E$=PS(INT(R/2)-2):: H$=INT(C/2)-6
880 SUBEND
840 SUB WHAT((ES),(H))
850 DISPLAY AT(20, 4): "Episthmi:" :: DISPLAY AT(21, 4): "Dyskolia:" ::
860 CALL SPRITE(#28, 160, 2, 32, 64)
867 CALL JOYST(1, X, Y)
868 CALL MOTION(#28, 0, 0, 5, 0'5' * (Y=4)-0'5' * (Y=4)-0'5' * (X=4)-0'5' * (X+4))
869 CALL KEY(1, RET, STA)
700 IF RET=16 THEN CALL LOC((ES),(H), ELSE=670
710 IF H=1 THEN 670 ELSE DISPLAY AT(20, 13): E$ :: DISPLAY AT(21, 13): STR$(H) :: SUB EXIT
720 SUBEND
730 SUB LOC((ES),(H))
770 DATA ASTRONOMIA, BIOLOGIA, GEVGRAFIA, FYSIKH, XHMEIA
780 CALL POSITION(#26, Y,X)
790 R=INT((Y+4)/8)+1:: C=INT((X+4)/8)+1
800 IF R=15 OR R<7 OR C>23 OR C<15 THEN H=0:: SUBEXIT
810 IF C=2=INT(C/2) OR R=2=INT(R/2) THEN H=0:: SUBEXIT
820 CALL MOTION(#28, 0, 0)
830 CALL SOUND(100, 1000, 0)
840 CALL DELSPRITE(#26)
850 CALL SPRITE(#28, 42, 7, R'8-12, C'1-8')
860 CALL MAGNIFY(2)
862 IF P=1 THEN 670
863 RESTORE 770
865 FOR I=1 TO 5:: READ PS(I)::* NEXT I::* P=1
870 E$=PS(INT(R/2)-2):: H$=INT(C/2)-6
880 SUBEND
890 SUB SCR1
900 DEF CTR$(WS)=SEG$("", 1,(28-LEN(WS))2) & WS
910 DISPLAY AT(1, 1): CTR$("Kokkinoi Gigantes")
920 DISPLAY AT(2, 1): CTR$("Asproi Nanoi")
930 DISPLAY AT(10, 1): CTR$("@ 1983 Diomhds Spinellhs")
940 DISPLAY AT(23, 1): CTR$("Pata Ena Koympi")
950 DISPLAY AT(24, 1): CTR$("Gia na Arxisei")
960 CALL KEY(5, RET, STA)::* IF STA=0 THEN 960
970 CALL CLEAR :: CALL SOUND(100, 1000, 0)
980 SUBEND
990 SUB ACC((N1$), (N2$), (N3$), (N4$))
1000 CALL CLEAR
1010 DISPLAY AT(4, 1): "Omada 1"
1020 ACCEPT AT(5, 1) VALIDATE(UALPHA) SIZE(10) : BEEP:N1$
1030 ACCEPT AT(6, 1) VALIDATE(UALPHA) SIZE(10) : BEEP:N2$
1040 DISPLAY AT(8, 1): "OMADA 2"
1050 ACCEPT AT(9, 1) VALIDATE(UALPHA) SIZE(10) : BEEP:N3$
1060 ACCEPT AT(10, 1) VALIDATE(UALPHA) SIZE(10) : BEEP:N4$
1070 CALL CLEAR
1080 SUBEND
1090 SUB MAIN((N1$), (N2$), (N3$), (N4$), (ES),(H),(W))
1100 CALL CLEAR :: CALL DELSPRITE(ALL) :: CALL CHAR((14), "181818181818181818181818FFFF1818") :: CALL HCHAR((9, 15, 95, 32) :: CALL VCHAR((15, 115, 141, 24)
1110 CALL VCHAR((9, 15, 142)
1120 DISPLAY AT(1, 4) SIZE(7): "OMADA 1"
1130 DISPLAY AT(1, 19) SIZE(7): "OMADA 2"
1140 DISPLAY AT(3, 2) SIZE(2): "1"
1150 DISPLAY AT(3, 3) SIZE(10): N1$
1160 DISPLAY AT(3, 16) SIZE(2): "1"
1170 DISPLAY AT(18) SIZE(10): N3$
1180 DISPLAY AT(4, 1) SIZE(2): "2"
1190 DISPLAY AT(4, 3) SIZE(10): N2$
Perseverance and discipline can get you a long way.
Write clear code, even when the environment doesn’t make it easy for you
procedure GenerateLabel(var Line : Linetype);
{Will remove the label from the line and add it to the symbol table}
begin
ConvertUppercase(Line);
If Copy(Line,1,1)<>' ' then {Label exists}
begin
  LabelName:=Copy(Copy(Line,1,Pos(' ',Line)-1),1,SymbolLength);
  CharacterizeSymbol(LabelName,PC,False,Relocatable,LabelS);
  {Labels become valid only after the directive is proved that it does no
  change their value i.e. it is not an EQU directive. This is done in the
  ObjectProcess procedure}
  Line:=Copy(Line,Pos(' ',Line),LineLength);
end
else
  LabelName:='$'; {For no label $ is implied so it is a nice place holder}
  While Copy(Line,1,1)=' ' do
    Line:=Copy(Line,2,LineLength);
end;
Structure your code into small units
Document your code with ample comments
Software development practices

1. Put source code under revision control
2. Perform frequent small commits
3. Follow the language’s style guide
4. Choose precise and consistent identifier names
5. Code in small units
6. Write unit tests
7. Separate concerns in module
8. Don’t repeat yourself
9. Ensure compliance through continuous integration
10. Release your code as open source software
Global Analysis and Transformations in Preprocessed Languages
Dionisis Spinellis, Mentor, IEEE

1 INTRODUCTION
Preprocessor program transformations are widely recog-
nized as a significant method for performing design
transformations and improving the quality of software.
Transformation tools are often used to improve the
sustainability and maintainability of programs.

2 WORK CONTEXT
Stencils transform text [3], and while in this paper,
we are interested in the context of the C++ compiler,
we believe that the same principles can be applied.

3 CONCLUSION
This paper introduces a new approach to
transformations in C++ programs, and we believe
that this approach will be useful in improving the
sustainability and maintainability of future
C++ programs.

Research
Optimizing header file include directives
Dionisis Spinellis*
Athens University of Economics and Business, Patission 76, GR-104 42 Athens, Greece

SUMMARY
A number of widely-used programming languages
are internally included files as a way to share and
reuse code. However, these files can be difficult
to maintain and can cause problems such as
linking order issues and conflicts. In this paper,
we propose a new approach to optimize the
include directives in these files.

*Correspondence to: Dionisis Spinellis, Athens University of Economics and Business, Patission 76, GR-104 42 Athens, Greece, d.spinellis@aegean.gr
Contact grant sponsor: European Community Sixth Framework Programme: Software Quality Observatory for Open Source Software (OQOSS); contract grant number: IST-2003-057311
Copyright © 2009 John Wiley & Sons, Ltd.

Build systems and tools
Keep in contact with industrial practice
Echoes from Space: Grouping Commands with Large-Scale Telemetry Data

Alexander Lattas
Department of Computing
Imperial College London
London, United Kingdom
alexander.lattas7@imperial.ac.uk

Diodimis Spinellis
Department of Management Science and Technology
Athens University of Economics and Business
Athens, Greece
dkdab@aegean.gr

ICSE’SEP ’18, May 27–June 2 2018, Gothenburg, Sweden

ABSTRACT
Background: As evolving desktop applications continuously add new features and grow more complex with deeper user interfaces and more sophisticated environments, it becomes inefficient to follow simple heuristics for grouping exec commands in multiple co-evolving deployment, trace-based software engineering studies on user performance prediction and command grouping optimization lack evidence-based answers on choosing a systematic grouping method.

Research Questions: We investigate the scope of command grouping in evolving desktop applications over time, through an average task completion time and improve their relative performance, as well as the benefit of using detailed interaction logs compared to sampling. Method: We introduce seven grouping methods and compare their performance based on extensive telemetry data, collected from a program run of a co-evolving application.

Results: We find that methods using global frequencies, user-specific frequencies, and ontologies achieve the best performance, with significant improvements compared to the baseline. Conclusion: We propose a new method, using the user’s average task completion time and improve their relative performance, as well as the benefit of using detailed interaction logs compared to sampling. Method: We introduce seven grouping methods and compare their performance based on extensive telemetry data, collected from a program run of a co-evolving application.

Results: We find that methods using global frequencies, user-specific frequencies, and ontologies achieve the best performance, with significant improvements compared to the baseline. Conclusion: We propose a new method, using the user’s average task completion time and improve their relative performance, as well as the benefit of using detailed interaction logs compared to sampling. Method: We introduce seven grouping methods and compare their performance based on extensive telemetry data, collected from a program run of a co-evolving application.

Results: We find that methods using global frequencies, user-specific frequencies, and ontologies achieve the best performance, with significant improvements compared to the baseline. Conclusion: We propose a new method, using the user’s average task completion time and improve their relative performance, as well as the benefit of using detailed interaction logs compared to sampling. Method: We introduce seven grouping methods and compare their performance based on extensive telemetry data, collected from a program run of a co-evolving application.

Results: We find that methods using global frequencies, user-specific frequencies, and ontologies achieve the best performance, with significant improvements compared to the baseline. Conclusion: We propose a new method, using the user’s average task completion time and improve their relative performance, as well as the benefit of using detailed interaction logs compared to sampling. Method: We introduce seven grouping methods and compare their performance based on extensive telemetry data, collected from a program run of a co-evolving application.

Results: We find that methods using global frequencies, user-specific frequencies, and ontologies achieve the best performance, with significant improvements compared to the baseline. Conclusion: We propose a new method, using the user’s average task completion time and improve their relative performance, as well as the benefit of using detailed interaction logs compared to sampling. Method: We introduce seven grouping methods and compare their performance based on extensive telemetry data, collected from a program run of a co-evolving application.

Results: We find that methods using global frequencies, user-specific frequencies, and ontologies achieve the best performance, with significant improvements compared to the baseline. Conclusion: We propose a new method, using the user’s average task completion time and improve their relative performance, as well as the benefit of using detailed interaction logs compared to sampling. Method: We introduce seven grouping methods and compare their performance based on extensive telemetry data, collected from a program run of a co-evolving application.

Results: We find that methods using global frequencies, user-specific frequencies, and ontologies achieve the best performance, with significant improvements compared to the baseline. Conclusion: We propose a new method, using the user’s average task completion time and improve their relative performance, as well as the benefit of using detailed interaction logs compared to sampling. Method: We introduce seven grouping methods and compare their performance based on extensive telemetry data, collected from a program run of a co-evolving application.

Results: We find that methods using global frequencies, user-specific frequencies, and ontologies achieve the best performance, with significant improvements compared to the baseline. Conclusion: We propose a new method, using the user’s average task completion time and improve their relative performance, as well as the benefit of using detailed interaction logs compared to sampling. Method: We introduce seven grouping methods and compare their performance based on extensive telemetry data, collected from a program run of a co-evolving application.

Results: We find that methods using global frequencies, user-specific frequencies, and ontologies achieve the best performance, with significant improvements compared to the baseline. Conclusion: We propose a new method, using the user’s average task completion time and improve their relative performance, as well as the benefit of using detailed interaction logs compared to sampling. Method: We introduce seven grouping methods and compare their performance based on extensive telemetry data, collected from a program run of a co-evolving application.

Results: We find that methods using global frequencies, user-specific frequencies, and ontologies achieve the best performance, with significant improvements compared to the baseline. Conclusion: We propose a new method, using the user’s average task completion time and improve their relative performance, as well as the benefit of using detailed interaction logs compared to sampling. Method: We introduce seven grouping methods and compare their performance based on extensive telemetry data, collected from a program run of a co-evolving application.

Results: We find that methods using global frequencies, user-specific frequencies, and ontologies achieve the best performance, with significant improvements compared to the baseline. Conclusion: We propose a new method, using the user’s average task completion time and improve their relative performance, as well as the benefit of using detailed interaction logs compared to sampling. Method: We introduce seven grouping methods and compare their performance based on extensive telemetry data, collected from a program run of a co-evolving application.

Results: We find that methods using global frequencies, user-specific frequencies, and ontologies achieve the best performance, with significant improvements compared to the baseline. Conclusion: We propose a new method, using the user’s average task completion time and improve their relative performance, as well as the benefit of using detailed interaction logs compared to sampling. Method: We introduce seven grouping methods and compare their performance based on extensive telemetry data, collected from a program run of a co-evolving application.

Results: We find that methods using global frequencies, user-specific frequencies, and ontologies achieve the best performance, with significant improvements compared to the baseline. Conclusion: We propose a new method, using the user’s average task completion time and improve their relative performance, as well as the benefit of using detailed interaction logs compared to sampling. Method: We introduce seven grouping methods and compare their performance based on extensive telemetry data, collected from a program run of a co-evolving application.

Results: We find that methods using global frequencies, user-specific frequencies, and ontologies achieve the best performance, with significant improvements compared to the baseline. Conclusion: We propose a new method, using the user’s average task completion time and improve their relative performance, as well as the benefit of using detailed interaction logs compared to sampling. Method: We introduce seven grouping methods and compare their performance based on extensive telemetry data, collected from a program run of a co-evolving application.

Results: We find that methods using global frequencies, user-specific frequencies, and ontologies achieve the best performance, with significant improvements compared to the baseline. Conclusion: We propose a new method, using the user’s average task completion time and improve their relative performance, as well as the benefit of using detailed interaction logs compared to sampling. Method: We introduce seven grouping methods and compare their performance based on extensive telemetry data, collected from a program run of a co-evolving application.

Results: We find that methods using global frequencies, user-specific frequencies, and ontologies achieve the best performance, with significant improvements compared to the baseline. Conclusion: We propose a new method, using the user’s average task completion time and improve their relative performance, as well as the benefit of using detailed interaction logs compared to sampling. Method: We introduce seven grouping methods and compare their performance based on extensive telemetry data, collected from a program run of a co-evolving application.

Results: We find that methods using global frequencies, user-specific frequencies, and ontologies achieve the best performance, with significant improvements compared to the baseline. Conclusion: We propose a new method, using the user’s average task completion time and improve their relative performance, as well as the benefit of using detailed interaction logs compared to sampling. Method: We introduce seven grouping methods and compare their performance based on extensive telemetry data, collected from a program run of a co-evolving application.

Results: We find that methods using global frequencies, user-specific frequencies, and ontologies achieve the best performance, with significant improvements compared to the baseline. Conclusion: We propose a new method, using the user’s average task completion time and improve their relative performance, as well as the benefit of using detailed interaction logs compared to sampling. Method: We introduce seven grouping methods and compare their performance based on extensive telemetry data, collected from a program run of a co-evolving application.

Results: We find that methods using global frequencies, user-specific frequencies, and ontologies achieve the best performance, with significant improvements compared to the baseline. Conclusion: We propose a new method, using the user’s average task completion time and improve their relative performance, as well as the benefit of using detailed interaction logs compared to sampling. Method: We introduce seven grouping methods and compare their performance based on extensive telemetry data, collected from a program run of a co-evolving application.

Results: We find that methods using global frequencies, user-specific frequencies, and ontologies achieve the best performance, with significant improvements compared to the baseline. Conclusion: We propose a new method, using the user’s average task completion time and improve their relative performance, as well as the benefit of using detailed interaction logs compared to sampling. Method: We introduce seven grouping methods and compare their performance based on extensive telemetry data, collected from a program run of a co-evolving application.

Results: We find that methods using global frequencies, user-specific frequencies, and ontologies achieve the best performance, with significant improvements compared to the baseline. Conclusion: We propose a new method, using the user’s average task completion time and improve their relative performance, as well as the benefit of using detailed interaction logs compared to sampling. Method: We introduce seven grouping methods and compare their performance based on extensive telemetry data, collected from a program run of a co-evolving application.

Results: We find that methods using global frequencies, user-specific frequencies, and ontologies achieve the best performance, with significant improvements compared to the baseline. Conclusion: We propose a new method, using the user’s average task completion time and improve their relative performance, as well as the benefit of using detailed interaction logs compared to sampling. Method: We introduce seven grouping methods and compare their performance based on extensive telemetry data, collected from a program run of a co-evolving application.
static void check_joints()
{
    register i, j;
    double maxx, minx, maxy, miny;
    int p1, p2, p3;
    double d, dx, dy, dz, a, b, c;
    int surfcount;
    /* zplane is true when the surface contains the z coordinate.
     * vert is true when a line from the surface is vertical
     * /
    int zplane;
    double zmin;
    struct surfacestruc *s;

    for( surfcount = 0 ; surfcount < surfacenum ; surfcount++ ){
        s = surfaces[surfcount];
        /* In partial views eliminated surfaces don't obscure */
        if( partial || window )
            for( i = 0 ; i < s->anglenum ; i++ )
                if( !(tag[s->joint[i]] & LINE ) )
                    goto nextsurface;
[80 more lines]
Assume responsibility for your learning
Fake it till you make it
A Dynamically Linkable Graphics Library

Dionysis D. Spinellis
Department of Computing, Imperial College of Science and Technology, 180 Queens Gate, London SW7 2BS, U.K.

SUMMARY

The design issues behind the implementation of an efficient and portable graphics library are discussed. A description of its components is given and the constraints leading to dynamic linking are presented. Techniques allowing the transparent dynamic linking of library elements are analyzed and two implementations of a system that automatically creates dynamically linkable code are presented. One implementation is based on traditional UNIX tools and the other on the perl programming language. The two implementations are compared.

KEY WORDS: Dynamic linking Graphics libraries Perl

INTRODUCTION

During the design of an interactive graphics pre- and post-processor for a finite element analysis system, the problem of portably displaying the output on a wide variety of graphics output devices was encountered. The program, initially, had to run on IBM-PC class machines running the MS-DOS operating system. In a latter stage it was ported to run under the UNIX operating system on Sun and microVAX workstations. The program is used to inspect structures represented by wire frames containing hundreds of elements in two distinct phases. First, before input to the finite element analysis program, the wire frame is examined in order to visually verify its form. After the analysis the program is used to inspect the distortions suffered under specific loads. The user may rotate the structure in three dimensions, view specific parts of it, label its joints and members and perform various other operations on it. The interactive nature of the program and the range of machines it was designed to operate on, made its design focus on a fast implementation. The main program consists of about 7000 lines of code written in the C[1] programming language.

MS-DOS does not provide an application graphics interface and the ROM Basic Input Output System (BIOS) [2] that is available on these machines does not support devices other than those manufactured by the machine vendor. In addition the functions it provides are minimal. Typical functions could display a character, set a point to a specified colour and set up the
A Dynamically Linkable Graphics Library

Diomidis D. Spinellis
Department of Computing, Imperial College of Science and Technology, 180 Queens Gate, London SW7 2BZ, U.K.

[INTRODUCTION]

During the design of an interactive graphics pre- and post-processor for a finite element analysis system, the problem of portably displaying the output on a wide variety of graphics output devices was encountered. The program, initially, had to run on IBM-PC class machines running the MS-DOS operating system. In a latter stage it was ported to run under the UNIX operating system on Sun and microVAX workstations. The program is used to inspect structures represented by wire frames containing hundreds of elements in two distinct phases.

[THE LIBRARY APPROACH]

[Functions provided]

No global variables are defined by the library. The functions provided rely on functions from the device specific library. The dichotomy of the two libraries was established gradually and in the early phases of the development functions tended to migrate...
Use computer tools to amplify your potential
The Elements of Computing Style

200+ Tips for Busy Knowledge Workers

Diomidis Spinellis
Mr. Dionisios Spinellis,

Dear Mr. Spinellis,

I have now received the referees' reports on your manuscript "A dynamically linkable graphics library".

These reports are enclosed.

In view of the referees' comments, I regret to say that I shall not be taking up your offer of this work for publication. With this letter, I am returning all the manuscript material that is presently in my files. Thank you, nevertheless, for considering "Software" as a possible medium for publication.

Yours sincerely,

J. A. Campbell

---

The idea of a dynamically linkable graphics library is nice but what is presented here is no solution.

The first part of the paper looks at the design of a graphics library and comes to the same conclusions as most implementors (e.g., NAG and NCAR) that it is necessary, to aid portability, to design a small device dependent set of primitives. The details given in Tables 1, 2 and 3 are unnecessary.

The author mentions (p 10) that "a portable windowing library is under consideration", it looks as though X-windows has beaten him to it!

The underlying theme of the paper is the need for portability and an easy solution to multiple devices (relinking is considered arduous and multiple executable modules difficult to maintain). The solution provided is operating system dependent (bottom p 12), 'highly compiler specific' (top p 13) (the code presented in this section even has embedded magic numbers!), linker dependent (p 14) and macro assembler dependent (p 19).

This 'solution' can only be described as a 'hack' which has been partially automated using the UNIX tools awk and sed. Apparently even the automation is ugly - 'Its interfaces are unclear and much of the work is done in a non-obvious and highly involved way!'

Finally the comparison with perl promised in the summary is very superficial and is compressed into under a page.

Continued .................
Publish often, be prepared to fail,  
... until you succeed
Unix PDP-11 Emulator
(A511 & Em11)
User’s Guide

Duncan White
Ann-Summer Pendley
Department of Elec

ABSTRACT

As11 and em11 form an emulated PDP-11* environment which can be used on
UNIVAC systems to design and develop simple PDP-11 assembly language programs. The
emulated PDP-11 includes 16K bytes of store, a screen, a keyboard, a line printer, and
two intermediate access disks.

1. Filename Conventions:

Just as Modula-2 uses standard suffixes such as .opl and .mod to identify files as belonging to Modula-2, so
the PDP-11 system uses the suffixes .a11 for an assembly language file and .a11 for an emulator input file.

2. The Assembler:

As11 is a free-form assembler, accepting all the standard PDP-11 mnemonics and operand types. It is
invoked by:

as11

The action of the assembler is to translate the single .a11 file named on the command line [you may omit
the .a11 suffix] into the corresponding .a1 file

Error messages and warnings during assembly are reported on the standard error stream. These are
intended to be self-explanatory.

The assembler continues after a warning, but stops after a fatal error.

* PDP-11 is a registered trademark of UNIVAC
* Unix is a registered trademark of AT&T Bell Labs

2.1. An Example Assembly Language Program

To make the following discussion clearer, here is a simple example of a PDP-11 Assembly Language pro-
gram.

; An Example PDP-11 Assembly Language Program

; A useful ASCII char. newline
nl  = 12

; Make space for the stack
org 560

; then declare the startup
org 1000

; start:

; initialise the stack, pc
mov $stack, sp

; pc, pc, str_msg
mov str_msg, (sp)

; byte / isn’t it a lovely day ?, nl nl
byte 0

; byte nl nl “Hello there everyone”
byte nl nl

; byte 0x68, 0x6c, 0x61, 0x72
byte 0

; byte 0
byte 0

; byte 0; byte 0
byte 0

; byte 0
byte 0

; byte
byte 0

; byte
byte 0

For the moment, let us not worry about the str_msg routine. Accept that it simply displays a null termi-
nated message whose starting address is stored on the stack.

2.2. The Format of Assembly Language Programs

Most of the lines in the above program contain a single PDP-11 instruction. Some lines, however, declare
labels, or perform assembler directives (known as pseudo-ops)

Any line may be terminated by a comment, introduced by a semi-colon which acts until the end of the cur-
rent line. A line, if so desired, can contain nothing except a comment.

Between the various constituents of a line, you may place any number of tabs and blanks which act as sep-
arators.

The assembler is not sensitive to upper and lower case.

2.3. Basic Concepts

2.3.1. Symbols

A symbol is the assembler equivalent of a Modula-2 constant. That is, it is a name which is used to repre-
sent a particular numeric value, increasing the readability of a program.

It is an error to redefine a symbol.

The assembler accepts indefinite-length symbols, which are sequences of alphanumeric and underscore
characters, where the first character is not numeric.
Partner with an experienced mentor
In the Department of Management Science and Technology at the Athens University of Economics and Business, current and former research and lab associates: Achilleas Angiostopoulos, Stefanos Androgioullis-Theotokis, Konstantinos Chatzimichalopoulos, Dimitris Andreadis, Vassilis Karakostas, Maria Keregha, Christos Lazarts, Dimitris Mitropoulos, Christos Oikonomou, Tushar Sharma, Sofoklis Stavrakis, Konstantinos Strobel, Vasileios Vlahos, and Giorgos Zouganis.


At the European Computer Industry Research Center (ECIRC): Mireille Ducassé, Anna-Maria Emde, Alexander Herald, Paul Martin, and Dave Morton.

At Imperial College London in the Department of Computer Science: Vasileios Capolaos, Mark Doetsch, Sophia Drossopoulou, Basile Dryden, Dave Edmonds, Susan Eisenback, Filippo Frangulis Anastasios Hadjicostis, Paul Kelly, Stephen J. Lacey, Phil Male, Lee M. J. McLaughlin, Stuart McRobert, Marelle Melichrimidis, Jan-Simon Pendry, Mark Taylor, Periklis Tsagkarios, and Duncan White.

In the Computer Science Research Group (CSRG) at the University of California at Berkeley: Keith Bostic.


Document everything you implement
Open Source Software Contributions

• Port Perl to MS-DOS
• Re-implement sed(1) for BSD Unix (now in macOS, FreeBSD)
• Trace tool for MS-DOS
• RCS utility functions
• zopen(3) compression interface
• NetPBM tools

www.spinellis.gr/sw/
Gains

• Fought boredom by working on challenging problems
• Honed coding skills
• Learned how to read/implement standards (POSIX)
• Networking (people)
• Established (a tiny) reputation
Contribute to / initiate open source software projects
Chapter 2

Related Work: Multiparadigm Programming

In this chapter we begin our exploration of the area of multiparadigm programming by examining the work that has been done up to now. Research in the area of multiparadigm programming can be divided into three different areas:

1. Programming paradigms: work in this area examines the notion of programming paradigms, their relationship to language design, and their effect on the software production environment.

2. Multiparadigm languages: during the literature research for this thesis we found more than 90 languages supporting more than one programming paradigm. Although not all languages were explicitly directed towards multiparadigm programming per se we believe that there were lessons to be learned from their collective study.

3. Multiparadigm programming frameworks: some researchers have come up with suitable abstractions and systems that support multiparadigm programming in general without targeting specific programming paradigms. Again in this case at least one of the systems covered was not the result of explicit multiparadigm research effort although it supports programming in multiple programming paradigms.

Our approach is geared towards producing a multiparadigm design methodology, a prototype system based on that methodology, and multiparadigm programming environment built upon that system. Therefore, the two last research areas are directly relevant to our research. We also examine the first area, because we believe that the notion of a programming paradigm is central to the theme of this thesis. Mixed language programming environments which only deal with languages based on a single paradigm (such as [Fis94]), and generic concurrent, distributed, heterogeneous systems, and module interface languages [Tiu92] that could potentially be used as multiparadigm frameworks (such as [Bay92]) are not examined. A thorough survey of distributed system languages can be found in [BST98], and of concurrent logic programming languages in [Shal99]; a set of articles on concurrent object-oriented programming can be found in [CAC93], and a survey of specific concurrent Smalltalk implementations in
### Table 2.4: Implementations combining the functional and logic paradigms

<table>
<thead>
<tr>
<th>Name</th>
<th>References</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALF</td>
<td>[Hao90, Hao91]</td>
<td>WAX extension</td>
</tr>
<tr>
<td>ALICE</td>
<td>[CS78, FT87]</td>
<td>ALM-interpretor on top of J-Lisp</td>
</tr>
<tr>
<td>Apple</td>
<td>[Col80, Col80]</td>
<td>ALM-interpretor in Prolog</td>
</tr>
<tr>
<td>BON8</td>
<td>[Bos87]</td>
<td>ALM-interpretor on Scheme</td>
</tr>
<tr>
<td>EPL</td>
<td>[JSG83, JS86]</td>
<td>Language</td>
</tr>
<tr>
<td>FGL-LV</td>
<td>[Liu85, Lii88]</td>
<td>Extensions to graph reduction language FGL-LV</td>
</tr>
<tr>
<td>FPL</td>
<td>[FPL82]</td>
<td>Extensions to TEL, functional language</td>
</tr>
<tr>
<td>Fresh</td>
<td>[Sno90]</td>
<td>Extensions to functional</td>
</tr>
<tr>
<td>FruM</td>
<td>[YZ96]</td>
<td>Interpreter implemented in Prolog</td>
</tr>
<tr>
<td>HASK</td>
<td>[Mee90]</td>
<td>Implemented in C-Prolog</td>
</tr>
<tr>
<td>HCCPRYR</td>
<td>[Che90]</td>
<td>Implemented on top of Lisp</td>
</tr>
<tr>
<td>HETE2</td>
<td>[HT92]</td>
<td>Extensions to Prolog</td>
</tr>
<tr>
<td>Ide90</td>
<td>[Ide90]</td>
<td>Theoretical framework</td>
</tr>
<tr>
<td>IdNov90</td>
<td>[IPW90]</td>
<td>Operational Semantics</td>
</tr>
<tr>
<td>LML</td>
<td>[BD94, BD96]</td>
<td>Extensions to functional</td>
</tr>
<tr>
<td>LOGLISP</td>
<td>[BS82]</td>
<td>Extensions to Lisp</td>
</tr>
<tr>
<td>Leaf</td>
<td>[BRLM86, BRLM85]</td>
<td>Plan for hardware implementation</td>
</tr>
<tr>
<td>Nai98</td>
<td>[Nai88]</td>
<td>Techniques</td>
</tr>
<tr>
<td>Quilt</td>
<td>[SS95a]</td>
<td>Implemented in Prolog as a translator to Prolog</td>
</tr>
<tr>
<td>SProlog</td>
<td>[Sno82]</td>
<td>Implemented on top of Prolog</td>
</tr>
<tr>
<td>SchemeLog</td>
<td>[Bos90]</td>
<td>Interpreter on Scheme</td>
</tr>
<tr>
<td>TABLOG</td>
<td>[NMW94, NMW95]</td>
<td>Language implemented in Lisp</td>
</tr>
<tr>
<td>TerniDesc</td>
<td>[Nik85]</td>
<td>Prolog extension</td>
</tr>
<tr>
<td>YS96</td>
<td>[YS96]</td>
<td>Semantic framework</td>
</tr>
</tbody>
</table>

### Table 2.5: Characteristics of functional and logic paradigm combinations

<table>
<thead>
<tr>
<th>Paradigm</th>
<th>Characteristics</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.PAK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C with Rule</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extensions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prolog</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Predicates</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logic in APL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shrewd</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 2.6: Implementations combining the functional and logic paradigms

<table>
<thead>
<tr>
<th>Name</th>
<th>References</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSM</td>
<td>[Ran97]</td>
<td>Extensions to C</td>
</tr>
<tr>
<td>Echidna</td>
<td>[HSS92]</td>
<td>Implemented on top of Lisp</td>
</tr>
<tr>
<td>Edoc2</td>
<td>[Bos86]</td>
<td>Prolog EB02</td>
</tr>
<tr>
<td>Enhanced C</td>
<td>[Kao83]</td>
<td>Compiler producing C</td>
</tr>
<tr>
<td>Fooplog</td>
<td>[GT97]</td>
<td>Language</td>
</tr>
<tr>
<td>Icem</td>
<td>[OGR97, Gr94]</td>
<td>Language</td>
</tr>
<tr>
<td>KE88</td>
<td>[KE88]</td>
<td>LOGOS and Prolog</td>
</tr>
<tr>
<td>Kaleidoscope</td>
<td>[FH92]</td>
<td>Language interpreter</td>
</tr>
<tr>
<td>Lex</td>
<td>[Lev75]</td>
<td>C preprocessor</td>
</tr>
<tr>
<td>MIL-Lex</td>
<td>[AMT99]</td>
<td>C preprocessor</td>
</tr>
<tr>
<td>MIL-Yoc</td>
<td>[TA90]</td>
<td>ML preprocessor</td>
</tr>
<tr>
<td>SPOOL</td>
<td>[FH96, Yok96]</td>
<td>Implemented on top of Prolog VM</td>
</tr>
<tr>
<td>Uniform</td>
<td>[Koch86]</td>
<td>Implemented on top of Lisp</td>
</tr>
<tr>
<td>Yacc</td>
<td>[Ad95]</td>
<td>C preprocessor</td>
</tr>
</tbody>
</table>

### Table 2.7: Characteristics of imperative and logic paradigm combinations

<table>
<thead>
<tr>
<th>Name</th>
<th>Characteristics</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modules-Prolog</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prolog</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Predicates</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C with files</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 2.8: Language characteristics

<table>
<thead>
<tr>
<th>Language Characteristics</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional</td>
<td></td>
</tr>
<tr>
<td>Imperative</td>
<td></td>
</tr>
<tr>
<td>Object-Oriented</td>
<td></td>
</tr>
<tr>
<td>Logic</td>
<td></td>
</tr>
<tr>
<td>Distributed</td>
<td></td>
</tr>
</tbody>
</table>

### Table 2.24: Number of languages for the common paradigm combinations

<table>
<thead>
<tr>
<th>Number of Languages</th>
<th>24</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>11</th>
<th>7</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
</table>
The facilities of a Prolog interpreter are provided to a Modula-2 programmer through a library. Predicates, that can be called from the Prolog interpreter, are written in Modula-2. The library includes term handling procedures.

<table>
<thead>
<tr>
<th>Program</th>
<th>Implementation</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>chars</td>
<td>Sh</td>
<td>13</td>
</tr>
<tr>
<td>chartabl</td>
<td>Perl</td>
<td>59</td>
</tr>
<tr>
<td>dbgrep</td>
<td>Perl</td>
<td>27</td>
</tr>
<tr>
<td>desclist</td>
<td>Perl</td>
<td>17</td>
</tr>
<tr>
<td>imptable</td>
<td>Perl</td>
<td>38</td>
</tr>
<tr>
<td>linesort</td>
<td>Perl</td>
<td>19</td>
</tr>
<tr>
<td>llinesor</td>
<td>Perl</td>
<td>26</td>
</tr>
<tr>
<td>maketext</td>
<td>Perl</td>
<td>117</td>
</tr>
<tr>
<td>pars</td>
<td>Sh</td>
<td>13</td>
</tr>
<tr>
<td>partable</td>
<td>Perl</td>
<td>27</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>Perl</strong></td>
<td><strong>697</strong></td>
</tr>
</tbody>
</table>
Automate data collection, analysis, presentation
Prefer building plumbing to porcelain
Write small tools that do one thing well
\[ \int x^2 \sin x \, dx = -x^2 \cos x + 2x \sin x + 2 \cos x + K \]

\[ \int_{1}^{0} \frac{1}{1+x^2} \, dx = 0.785398 + 0.001 \]

<table>
<thead>
<tr>
<th>Function</th>
<th>Paradigm</th>
<th>Module</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbolic integration</td>
<td>btrack</td>
<td>sint.pb</td>
<td>127</td>
</tr>
<tr>
<td>Lexical analysis</td>
<td>regex</td>
<td>scan.pl</td>
<td>47</td>
</tr>
<tr>
<td>Expression parsing</td>
<td>bnf</td>
<td>parse.py</td>
<td>76</td>
</tr>
<tr>
<td>Numeric integration</td>
<td>fun</td>
<td>aint.pl</td>
<td>75</td>
</tr>
<tr>
<td>Interfacing</td>
<td>term</td>
<td>ui.pt</td>
<td>131</td>
</tr>
<tr>
<td>Graph creation</td>
<td>imper</td>
<td>main.c</td>
<td>51</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>blueprint</td>
<td></td>
<td>507</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Paradigm</th>
<th>PDF</th>
<th>imper</th>
<th>bnf</th>
<th>regex</th>
<th>term</th>
<th>fun</th>
<th>Total</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>imper</td>
<td>43</td>
<td>1192</td>
<td>119</td>
<td>84</td>
<td>666</td>
<td>554</td>
<td>2269</td>
<td>53.3</td>
</tr>
<tr>
<td>term</td>
<td>70</td>
<td></td>
<td>316</td>
<td></td>
<td>237</td>
<td>43</td>
<td>840</td>
<td>19.7</td>
</tr>
<tr>
<td>btrack</td>
<td>60</td>
<td>305</td>
<td>59</td>
<td></td>
<td></td>
<td></td>
<td>405</td>
<td>9.5</td>
</tr>
<tr>
<td>fun</td>
<td>140</td>
<td>424</td>
<td>143</td>
<td>43</td>
<td>121</td>
<td>43</td>
<td>4259</td>
<td>100.0</td>
</tr>
<tr>
<td>bnf</td>
<td>95</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>regex</td>
<td>379</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>787</td>
<td>1192</td>
<td>424</td>
<td>143</td>
<td>1219</td>
<td>43</td>
<td>4259</td>
<td>100.0</td>
</tr>
<tr>
<td>%</td>
<td>18.5</td>
<td>28.0</td>
<td>10.0</td>
<td>3.4</td>
<td>28.6</td>
<td>1.0</td>
<td>100.0</td>
<td></td>
</tr>
</tbody>
</table>
Play on your strengths
III. IDENTIFICATION COMPLEXITY

A. A Very Briefly Defined (1) to Access a Type of Machine

B. A Machine in the Form Defined (1) to Access a Type of Machine

IV. SOFTWARE

A. Categorically: Creditor Software (C) — when issued to

B. A Type of Machine for a Type of Machine

C. A Machine in the Form Defined (1) to Access a Type of Machine

D. A Machine in the Form Defined (1) to Access a Type of Machine

E. A Machine in the Form Defined (1) to Access a Type of Machine

F. A Machine in the Form Defined (1) to Access a Type of Machine

V. CONCLUSION

A. The Very Briefly Defined (1) to Access a Type of Machine

B. A Machine in the Form Defined (1) to Access a Type of Machine

C. A Machine in the Form Defined (1) to Access a Type of Machine

D. A Machine in the Form Defined (1) to Access a Type of Machine
A Simulated Annealing Approach for Buffer Allocation in Reliable Production Lines

Dionisios D. Spinellis a,*, Chrissoleon T. Papadopoulos b

a Department of Mathematics, GR-812 00 Kortsoni, University of the Aegean, Greece
E-mail: depma@aegean.gr

b Department of Business Administration, GR-812 00 Chios Island, University of the Aegean, Greece
E-mail: lps@aegean.gr

We describe a simulated annealing approach for solving the buffer allocation problem in reliable production lines. The problem entails the determination of near optimal buffer allocation plans in large production lines with the objective of maximizing their average throughput. The latter is calculated utilizing a decomposition method. The allocation plan is calculated subject to a given amount of total buffer slots in a computationally efficient way.

Keywords: Simulated annealing, production lines, buffer allocation, decomposition method

1. Introduction and Literature Review

Buffer allocation is a major optimization problem faced by manufacturing systems designers. It has to do with devising an allocation plan for distributing a certain amount of buffer space among the intermediate buffers of a production line. This is a very complex task that must account for the random fluctuations in mean production rates of the individual workstations of the lines. To solve this problem there is a need of two different tools. The first is a tool that calculates the performance measure of the line which has to be optimized (e.g., the average throughput or the mean work-in-process).

* This is a machine-readable rendering of a working paper draft that led to a publication. The publication should always be cited in preference to this draft using the reference in the previous footnote. This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author’s copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder.

** Corresponding author.

\[
\binom{N + K - 2}{K - 2} = \frac{(N + 1)(N + 2) \cdots (N + K - 2)}{(K - 2)!}
\]
Unix tools as visual programming components in a GUI-builder environment

Diodiris Spinellis 1,6

Department of Management Science and Technology, Athens University of Economics and Business, Patission 76, Athens GR-10434, Greece

SUMMARY

Development environments based on ActiveX controls and JavaBeans are marketed as ‘visual programming’ platforms; in practice their visual dimension is limited to the design and implementation of an application’s graphical user interface (GUI). The availability of sophisticated GUI development environments and visual component development frameworks is now providing viable platforms for implementing visual programming within general-purpose platforms, i.e., for the specification of non-GUI program functionality using visual representations. We describe how specially designed reflective components can be used in an industry-standard visual programming environment to semantically specify sophisticated data transformation pipelines that interact with GUI elements. The components are based on Unix-style filters repackaged as ActiveX controls. Their visual layout on the development environment canvas is used to specify the connection topology of the resultant pipeline. The process of converting filter-style programs into visual controls is automated using a domain-specific language. We demonstrate the approach through the design and the visual implementation of a GUI-based spell-checker. Copyright © 2001 John Wiley & Sons, Ltd.

KEY WORDS: visual programming; components; reflection; Unix tools; pipe and filter architecture; reuse

1. INTRODUCTION

A number of environments support the visual composition of graphical user interfaces (GUIs) using components with a predefined set of interfaces. In addition, technologies such as ActiveX and JavaBeans allow the development of visual components (typically GUI elements) that can be seamlessly incorporated into an integrated development environment (IDE) and subsequently used in application development. In this article we present how visual IDEs and components can be extended beyond GUI development to support visual programming for a particular domain.

1Correspondence to: Diodiris Spinellis, Department of Management Science and Technology, Athens University of Economics and Business, Patission 76, Athens GR-10434, Greece. E-mail: ds@athab.gr

Copyright © 2001 John Wiley & Sons, Ltd.

Received 10 November 2000
Revised 4 June 2001 and 17 September 2001
Accepted 16 September 2001
Make the most out of the time you have
1. Put the document under version control
2. Write readable and maintainable LaTeX source code
3. Avoid explicit formatting
4. Automate the management of bibliographic references
5. Use symbolic references
6. Automate the document's build
7. Use Continuous Integration
8. Use third-party LaTeX packages
9. Use style files
10. Learn how to set text, mathematics, tables, figures, and floats
Obtaining metrics from large code bodies is difficult for technical and operational reasons.\cite{Moc09,GS13}. On the technical side, code dependencies make it difficult to establish the full context needed in order to parse and semantically analyse the code. This is especially true for C code, where the compilation depends on system header files, compiler-defined macros, search paths, and compile-time flags passed through the build process.\cite{Spi03r,LKA11,GG12} The operational reasons are associated with the required throughput, though due to the relatively small number of releases we examined, this was not a major issue in this study.
The Decay and Failures of Web References

By Diomidis Spinellis

The widespread adoption of the Web as a mechanism for delivering information has brought with it the corresponding explosion of URL addresses and citations. URLs regularly appear on webpages, blogposts, business cards, press releases, and more. But the very nature of the Web and the way it is used has led to a situation where it is becoming increasingly difficult to ensure that any given URL will indeed continue to point to the same content in the future. This is not just an academic exercise: many organizations are now relying on the稳定性和可用性 of the Web as a critical aspect of their operations. The problem is particularly acute for organizations that are using the Web as a central repository of information, such as universities, hospitals, and government agencies. In this context, the ability to ensure that any given URL will continue to point to the same content is critical for the functioning of these organizations.


By Diomidis Spinellis

The Unix operating system was developed in the early 1970s at Bell Labs, and has since become a cornerstone of modern computing. It is a complex and highly modular system, with a rich set of tools and utilities that are used by programmers and system administrators alike. In this study, we examined the evolution of C programming practices in the Unix operating system over a period of 42 years, from 1973 to 2015. We found that there were significant changes in the way that C was used, with a shift towards more modular and reusable code, and an increased emphasis on testing and quality assurance.

The Introduction to C Programming

By Diomidis Spinellis

C is a high-level programming language that is commonly used for system programming, and is a standard feature of Unix, Solaris, Linux, and other Unix-like operating systems. It is a compiled language, and programs written in C are compiled into machine code that can then be executed directly by the computer's processor. This makes C a very powerful language, but it also means that programs written in C can be complex and difficult to understand. In this course, we will cover the basics of C programming, including data types, control structures, and functions. We will also cover more advanced topics, such as pointers, memory management, and error handling.
Create / collect your own data sets
The information furnace: consolidated home control

Abstract The Information Furnace is a basement-installed PC-type device that integrates existing consumer home-control, infotainment, security and communication technologies to transparently provide accessible and value-added services. A modern home contains a large number of sophisticated devices and technologies. Access to these devices is currently provided through a wide variety of disparate interfaces. As a result, end users face a bewildering array of confusing user-interfaces, access modes and price structures. In addition, as most devices function in isolation, important opportunities to exploit synergies between their functionalities are lost. The information furnace distributes data, provides services, and controls an apartment’s digital devices. Emphasis is placed on accessibility and on exploring the synergies that inevitably come up when these technologies and services are housed under a single roof. The prototype implementation I outline integrates on a FreeBSD server the distribution of MP3-encoded music to DNARD/NetBSD thin clients, an answering machine, a burglar alarm, an Internet router, a fax server, a backup server, and intelligent control of a PBX.

Keywords Automation - Consumer electronics - Home-control - Multi-modal interfaces

1 Introduction

Although our complex lives are not necessarily improved by each new technological widget we adopt, uncooperative devices and appliances with deficient user-interfaces can certainly conspire to frustrate us. Over the past three years I have experimented with a number of technologies that gave birth to the information furnace concept: a basement-installed PC-type device that integrates existing consumer home-control, infotainment, security, and communication technologies to transparently provide ubiquitous access and synergistic value-added services. In the following sections we will examine the devices and appliances lurking in the modern home, overview the problems associated with the current breed of devices, and go over the basic elements of the information furnace concept and its prototype implementation. Further implementation details on technologies behind the system we describe can be found in Spinellis [1]; this paper focuses on the system’s concept, architecture, and evaluation.

2 The modern home

A modern home contains a large number of sophisticated devices and technologies. Current and near future technologies and respective devices can be roughly categorised into the categories of home control, infotainment, security, communication and special-purpose devices.

2.1 Home control

Contemporary central heating systems are regulated by one external and a number of internal temperature sensors in conjunction with a control unit occupants use to set the desired room temperature. The system compares the internal room temperature to the setting of the control unit and, using the external temperature as a compensating factor, regulates the temperature of the water produced by the local heat-generating plant or the valve bringing remotely-heated water into the home. Burners often have their own control circuits based on target temperatures for the burner and the circulating pump, but we can regard them as a black box for the purposes of this article. Convenience elements associated
Position-Annotated Photographs: A Geotemporal Web

The GTWeb system exploits the synergies of integrating different information appliances and pose-locable data sources to create and present trip stories.

With the advent of digital cameras, photographs are a larger pool of data. However, in addition to their aesthetic and sentimental value, photos can also provide a wealth of information about the physical world. The GTWeb system, described in this paper, allows users to annotate photographs with metadata such as location, date, and time. This information can then be used to create a geotemporal web, which is a network of photographs that are linked to each other based on their spatial and temporal relationships.

Functional design

GTWeb is designed to be easy to use and to provide a seamless experience for users. The system includes a user interface that allows users to search for and view photographs, as well as to add annotations and links to other photographs. The system also includes a visualizer that allows users to explore the geotemporal web in a more intuitive way.

Figure 4. Example of a GTWeb interface, showing a timeline of photographs.

Conclusion

In conclusion, the GTWeb system provides a powerful tool for creating and sharing geotemporal web content. By integrating information from different sources, the system allows users to create rich, multimedia stories that can be shared with others. The system is easy to use and has a wide range of applications, from personal use to professional use.

Acknowledgments

This work was supported by the National Science Foundation under Grant No. 0650865. We would like to thank the anonymous reviewers for their valuable feedback.

References


Appendix

Figure 5. Example of a GTWeb timeline, showing a sequence of photographs.

Figure 6. Example of a GTWeb timeline, showing a sequence of photographs.
The Antikythera Mechanism: A Computer Science Perspective

In 1900, a group of sponge divers working off the Kythera island in the Mediterranean Sea discovered an ancient Greek mechanism. The discovery was initially ignored by the excavators, but the mechanism was later extensively researched and is now known as the Antikythera Mechanism.

The mechanism is a complex device that was used to predict the movements of the Sun, Moon, and planets. The mechanism is believed to be the earliest known device capable of simulating the movements of celestial bodies. It is estimated that the device was built around 200 BCE.

The Antikythera Mechanism is a mechanical computer that contains a helical gear system, which is driven by a series of gears and pulleys. The gears are arranged in a series of nested rings, which move in unison to predict the positions of the celestial bodies.

The mechanism is a testament to the advanced knowledge of the ancient Greeks in the field of astronomy. The device is believed to have been used to predict the positions of the Sun, Moon, and planets, which were of great importance to the ancient Greeks.

The Antikythera Mechanism is a significant archaeological discovery, and it provides an insight into the advanced technology of the ancient Greeks. It is a reminder of the importance of preserving and studying ancient artifacts to gain a better understanding of our past.
RESOURCES

A DIY LEGO CONTROLLER
A LOW-COST WAY TO PROGRAM LEGO MACHINES

needed was integration, configuration, and some glue software.

First, I needed to build an infrared control link, which is basically two infrared LEDs operated in the Raspberry Pi's general-purpose input/output (GPIO) connector and LEGO's receiver. I used schematics and instructions by Alex Bain to build the hardware. For the software, I downloaded and installed LIRC, a package that has support for decoding and transmitting signals used by over 2,500 different infrared remote controls.

Getting the LIRC package to work with my home-brew infrared led was a simple matter of editing some configuration files and specifying the GPIO pins I used for input and output.

Now I needed to get LIRC to send valid LEGO command signals. This means specifying the waveform—a pattern of infrared pulses—that must be sent for each LEGO command. Fortunately, LEGO has released a document specifying the protocol and format of all commands (for example, a binary value of 1 is transmitted by a pulse of 0.5 milliamp at a frequency of 38 kilohertz). Using this, I wrote a wrapper script in Python that would allow me to send the commands using the LIRC API. The script would also allow me to control the LEGO Power Functions motors and lights using the same interface.

Some local modifications were required to ensure compatibility with the LEGO software. I had to modify the software to work with the Raspberry Pi's GPIO interface, which is different from the LEGO's SPI interface. I also had to modify the LEGO software to work with the Raspberry Pi's GPIO interface, which is different from the LEGO's SPI interface.

Finally, I wrote a Scratch script that reads data from the LEGO Power Functions controller and sends it to the LEGO Power Functions command line client to be sent to the LEGO Power Functions server on the Raspberry Pi. The script connects to the LEGO server using SSH and sends the commands using the LEGO Power Functions command line client.

The final step was to test the LEGO command line client in a real environment. I enabled "remote sensor connections" in Scratch, which allows Scratch to communicate with the LEGO Power Functions server on the Raspberry Pi. The client software can be run on any computer with a Raspberry Pi connected to it, and it can be controlled using the LEGO Power Functions command line client.

This is a great way to get started with LEGO programmable robotics, and it shows how Scratch can be used to control LEGO motors and lights using the LEGO Power Functions API. With some additional work, this could be extended to control other LEGO components, such as sensors and actuators, using Scratch and the LEGO Power Functions API.

I hope this article has given you some ideas for using Scratch and LEGO to control programmable LEGO bricks. With some creativity and a bit of coding, you can create your own LEGO robots and machines using Scratch and LEGO Power Functions!
Takeaway?
Be curious, have fun!
Thank you!

dds@aueb.gr
www.spinellis.gr
@CoolSWEng
github.com/dspinellis
Image Credits

• HF65 instructions: Bruker
• HF65 image: DL4CS German Amateur Radio Station
• Sharp PC-1211: Denisfo - Own work CC BY-SA 3.0
• TI-99/4A: Rama & Musée Bolo - Own work CC BY-SA 2.0 fr
• IBM PC: Rama & Musée Bolo - Own work CC BY-SA 2.0 fr
• IBM Portable PC: Hubert Berberich (HubiB) CC BY-SA 3.0
• Τεχνική Εκλογή magazine: salax54


• Duncan White, Jan-Simon Pendry, and Diomidis Spinellis. Unix PDP-11 emulator (as11 & em11) user’s guide. Laboratory documentation, Imperial College, London, UK, January 1989.


www.spinellis.gr/pubs/