Scripting Runtime
Dynamic Analysis

Jonathan Cook
WODA 2003
The Point

• What about ad-hoc and one-shot analyses?
• Need to lower the development cost
  – instrumentation cost
  – development effort
• Scripting languages offer an ideal platform for providing these things
Warning!!!

• I LOVE scripting languages

• SL put the fun back into programming

• Able to see exciting results quickly
Scripting Languages

• Originally meant “shell scripts”
• Has evolved into programming that is
  – easy to use
  – less formal?
  – provides lots of built in functionality
• Perl, Tcl, Python, VB, PHP, …
RT Dynamic Analysis Needs:

- System under observation
- Instrumentation
- Frameworks (tricky stuff!)
- "Old-fashioned" programming
- Runtime dynamic analysis
- Instrumentation
- System under observation
Instrumentation

- Always going to be “hard”? 
- If done well, reusable 
- An ideal arena for systems programming languages
Programming

• Many different needs
  – high execution speed
  – high robustness
  – low development effort
  – high ease of use
  – low entry barrier

• Scripting languages can often provide the last three
Scripting Has:

• Extensibility
  – can define new commands in the language through conventional programming

• GUI building
  – easy, flexible, and dynamic GUI capabilities

• Event handling
  – both system and user-defined events
DA Example in Tcl

```tcl
proc mymalloc_begin {size} {
    global NumMallocs BytesMalloced AvgBlockSize CurrentAlloc
   incr NumMallocs
    incr BytesMalloced $size
    incr CurrentAlloc $size
    Setline $size red
    set AvgBlockSize [expr $BytesMalloced/$NumMallocs]
}

proc mymalloc_end {ptr size} {
    global MBlocks
    set MBlocks($ptr) $size
}
```
proc myfree_begin {ptr} {
    global NumFrees BytesFreed MBlocks CurrentAlloc
    incr NumFrees
    if {![info exists MBlocks($ptr)]} {
        puts stderr "Free error: block at [format "%x" $ptr] does not exist!"
        return
    }
    if {$MBlocks($ptr) < 0} {
        puts stderr "Free error: block at [format "%x" $ptr] already freed!"
        return
    }
    incr BytesFreed $MBlocks($ptr)
    incr CurrentAlloc [expr -1 * $MBlocks($ptr)]
    Setline $MBlocks($ptr) green
    set MBlocks($ptr) [expr -1 * $MBlocks($ptr)]
}
Instrumentation Level

- Interface from executable to Tcl is done in C
  - dynamic linker mods
  - wrapper generation from prototypes
  - future enhancements (data access, …)
void * mymalloc (size_t numbytes) {
    Tcl_Obj *cmdvector[3];
    void * retval;
    do_redirect = 0;
    cmdvector[0] = Tcl_NewStringObj("mymalloc_begin",-1);
    cmdvector[1] = Tcl_NewLongObj(numbytes);
    Tcl_EvalObjv(TclInterp,2,cmdvector,TCL_EVAL_GLOBAL);
    retval = malloc (numbytes);
    cmdvector[0] = Tcl_NewStringObj("mymalloc_end",-1);
    cmdvector[1] = Tcl_NewLongObj(retval);
    cmdvector[2] = Tcl_NewLongObj(numbytes);
    Tcl_EvalObjv(TclInterp,3,cmdvector,TCL_EVAL_GLOBAL);
    do_redirect = 1;
    return retval;
}
My Claims

- Scripting languages offer large code reuse base
  - avoid wheel reinvention
- Focus on language improves instrumentation interface design
  - what commands/events do I want to add?
  - what should their arguments be?
  - what options should they take?
  - when should control be transferred?
Scripting Needs:

- For dynamic analysis, some holes exist
- Event processing
  - event pattern triggers rather than just simple events
- Parsing
  - perhaps something like DCG’s
- Logical inference capability
  - embed some minimal WAM?
Reasons for Failure

• Slow!
  – dynamic analysis is often compute intensive

• Error-prone?
  – weak or no typing can lead to surprising errors

• Too many choices
  – how to select a scripting language to use?
Reasons for Success

• Scripting languages provide an incredibly rich context for programming
• Ideal foundation for building prototypes and ad-hoc analyses
• Interfacing to a scripting languages forces good design of instrumentation interface