Supporting PHP Dynamic Analysis in PHP AiR

Mark Hills

13th International Workshop on Dynamic Analysis (WODA 2015)
October 26, 2015
Pittsburgh, Pennsylvania, USA

http://www.rascal-mpl.org
PHP AiR: PHP Analysis in Rascal

• PHP AiR: a framework for PHP source code analysis

• Domains:
  • Program analysis (static/dynamic)
  • Software metrics
  • Empirical software engineering
PHP AiR: PHP Analysis in Rascal

• PHP AiR: a framework for PHP source code analysis

• Domains:
  • Program analysis (static/dynamic)
  • Software metrics
  • Empirical software engineering
PHP AiR: PHP Analysis in Rascal

• PHP AiR: a framework for PHP source code analysis

• Domains:
  • Program analysis (static/dynamic)
  • Software metrics
  • Empirical software engineering
A quick note on Rascal

• “Rascal is a domain specific language for source code analysis and manipulation a.k.a. meta-programming.” (http://www.rascal-mpl.org/)

• Language focus: program analysis, program transformation, domain-specific language creation

• Current projects across large numbers of domains, both outside and within academia (including this one!)

• Open source, committers worldwide
Original motivation: dynamic invocations

• Reflective capability in PHP for invoking functions and methods

• Runtime target given as PHP `callable`, not as regular identifier
  • Function name
  • Object instance and method name
  • Class name and method name
  • Closures (in newer versions, not common yet)

• Parameters passed as var-args or as array
Dynamic invocations: two quick examples

// From MediaWiki 1.19.1
if ($this->mPage->getID() != $this->mRev->getPage())
{
    $fun = array(get_class($this->mPage),'newFromID');
    $this->mPage =
        call_user_func($fun, $this->mRev->getPage());
}

// From WordPress 3.4
$args = wp_list_widget_controls_dynamic_sidebar(
    array(0 => $args,
          1 => $widget['params'][0]));
call_user_func_array('wp_widget_control', $args);
What are they used for? Why study them?

- Often used for plugin systems and user extensions
- Presence makes it hard to analyze the program
  - How do we build a call graph?
  - How do we compute types? aliases? taint?
- Indirection also slows execution (observational, no figures yet)
- Not uncommon, so cannot just ignore: 94 in WordPress 3.4, 149 in MediaWiki 1.19.1 (see our ISSTA 2013 paper for details)
Possible solution: code specialization

- Idea based on work by Furr, An, and Foster: *Profile-Guided Static Typing for Dynamic Scripting Languages* (OOPSLA 2009)
  
  - Trace executions of system, execute using test scripts
  
  - Replace dynamic features with static variants and “catch-all”

- Original work used Ruby, Mulder applied technique to PHP and WordPress (see thesis *Reducing Dynamic Feature Usage in PHP Code*)

- Results installation-specific, based on specific plugins used
Why not just use the existing solution?

• Earlier work had a complex tool chain, hard to set up and reuse

• Very scenario-specific, targeted specifically at dynamic invocations, we need a generic tracing framework

• No support for figuring out where strings come from, useful for analysis and empirical studies
Supporting dynamic analysis in PHP AiR

• Now: Support function trace analysis directly in PHP AiR
  
  • Flexible parsing and filtering capabilities
  
  • Directly in Rascal, easy to extend, share, replicate

• Future: support execution of tests from within Rascal
  
  • Initial support for driving xdebug exists, needs further work

• Early stage: Instrument interpreter to track origins of strings
Function trace analysis in action: PHP defines

• First, generate trace/traces (currently outside of PHP AiR)

• Parsing, stage 1: Read in line from trace file, determine the record type, apply initial filtering

• Parsing, stage 2: parse function parameters, apply additional filtering

• Major bottleneck is speed of parser

• Further challenge: location information in not precise, line-based
String origins

- Based on origin tracking (van Deursen, Klint, and Tip) and string origins (Inostroza, van der Storm, and Erdweg)

- Goal: figure out where strings come from, track transformations of strings through program execution

- Origins tracked using source locations of literals, info on external inputs, transformation functions; origin types based on how string is created

- Still very early in implementation
String origins: challenges

• PHPs main goal in life: generate strings

• String-handling code is often optimized, we need to undo this

• Looking into HHVM, changes may be less disruptive, Quercus may be an implementation dead-end (supports PHP 5.4, nothing newer)

• Also looking into K, instrument existing PHP semantics (e.g., *An Executable Formal Semantics of PHP* by Filaretti & Maffeis, ECOOP 2014), simplicity of interpreter may allow us to be more precise
Summary

• Dynamic analysis for PHP is needed to properly analyze and study dynamic language features

• We are extending PHP AiR to enable flexible dynamic analysis for PHP

• Trace parsing and filtering works well, adapting to handle undocumented xdebug outputs

• String origins work is still ongoing, reevaluating choice of platform, looking for interested students
Discussion

Thank you!
Any Questions?

• Rascal: http://www.rascal-mpl.org

• Me: http://www.cs.ecu.edu/hillsma