Back-in-Time Debugging in Heterogenous Software Stacks

Arian Treffer
arian.treffer@hpi.de

International Workshop on Program Debugging
October 23, 2017

Enterprise Platform and Integration Concepts group
Prof. Plattner, Dr. Uflacker
A Brief History of Program Debugging

- **1950**: Bug Killing
- **1960**: Data Dump Debugging
- **1970**: Dynamic Debugging
- **1980**: Command Line Debuggers
- **1990**: Symbolic Debuggers
- **2000**: Reversible Debuggers
- **2010**: Omniscient Debugging
- **2020**: Reproducible Fault Debugging

**ODB in the Database**
Debugging Enterprise Applications

- Concurrent users on one instance
- Distributed physical systems
- Often with central server
- Independent components
  - Different environments, languages
1. Motivation
2. Developers Needs
3. Prototype Implementation
4. Conclusion
Developer Interviews

5 SAP Developers, 4 Software Projects
- Liquidity risk management
- Customer analytics
- Predictive analytics for sales
- Point-of-sales explorer KPI dashboard
3-Tier Applications

Then

- Dumb Terminal
  - Business Logic
    - Data Analysis
      - (ORM)
  - Storage
  - (Templates)
  - TCP

Now

- Rich GUI
  - (HTML, JS)
  - HTTP
  - ODATA
  - Business Logic
    - Data Analysis
      - (In-memory, Distributed)
  - Request Handling, File Hosting
  - SQL, ...

Real-world Examples

Project 1

- Rich GUI (HTML, JS)
  - HTTP
  - ODATA
- Request Handling, File Hosting
  - SQL
- Business Logic Data Analysis (In-memory, Distributed)

55%

18%

27%

Project 2

90%

5%

5%

(estimated)
Debugging 3-Tier Applications

- Agile processes
  - No UI/database team
  - Debugging full stack

- Top-Down approach preferred
  - Find errors in request patterns, parameters, or responses
  - Then debug responsible sub-system
  - Switching between sub-systems is very time consuming
Prototype Debugger
Assumption:
- Omniscient debugging available for each sub-system

“Execution”
- Function call triggered by external event
- Independent from other executions

“Full execution history”
- Set of all execution transitively caused by root event
Identifying Sub-System Interactions

- Restore request/response relations
- Challenge: uniquely identify each execution
  - URL parameters, query string
  - Add unique argument
- Challenge: find request submissions
  - Well-known method calls
  - Some code-analysis required
- Challenge: find asynchronous response handlers
- No generalizable solutions (?)
High-level Visualizations

- Support top-down approach
Support top-down approach
Developer Feedback

- Visualizations
  - help to understand structure
  - Simplify switching between components

- Timing might be relevant
- Restarting executions is much easier now

- All three can be achieved without full tracing
Future Work

- Effective dynamic analysis across system boundaries
- Example:
  - Object → JSON String → HTTP Response → ... → Object
- Abstraction levels to avoid analyzing parsing code
Summary

1. Motivation
   - ODB for many systems
   - What about complex applications

2. Developers Needs
   - 3-tier application changes
   - Top-down debugging

3. Prototype Implementation
   - Request/response matching
   - Visualizations
   - Generalization

4. Conclusion
   - Low-hanging fruits
   - Future Work