Overview

1. Introduction
2. TIDE
3. Experience
4. Demonstration
5. Conclusions
TIDE is a generic debugging framework, used
  - to construct debuggers for domain specific languages
  - to debug heterogeneous and distributed applications
Motivation for TIDE

- TIDE is a generic debugging framework, used
  - to construct debuggers for domain specific languages
  - to debug heterogeneous and distributed applications
- It is a component of The Meta-Environment
  - http://www.meta-environment.org
  - source analysis and transformation from high level specifications
  - integration into a language independent IDE
  - meta-environment is a component-based heterogeneous application
Motivation for this talk

Interest for debugging at IBM, e.g.
- Eclipse IMP - IDE Meta Tooling Platform
- X10 - language for parallel programming
- PTP - parallel tools platform

In this talk
- the design of TIDE
- experience report
- demonstration of a nested debugging session
Related work

- “The design of a high-level, language-independent symbolic debugging system”, Johnson .......................... 1977
- “Debugging Distributed Applications Using a Coordination Architecture.”, Olivier .............................. 1997
- DDF: Eclipse Domain Specific Debugging Platform, Wu . 2007

Advertisements

- LDTA — Language Descriptions, Tools & Applications 2009
- EST — Special issues on Experimental Software and Toolkits (Elsevier)
Abstract Debugging Model

A debugger observes “primitive events”, and
- clusters, filters and reports the events
- influences the run-time (i.e. break and resume)
- queries the run-time (i.e. current point of execution)
Abstract Debugging Model

- A debugger observes “primitive events”, and
  - clusters, filters and reports the events
  - influences the run-time (i.e. break and resume)
  - queries the run-time (i.e. current point of execution)

- TIDE event rules guide debugging processes:
  - A port is a cluster of debugging events
  - A condition filters uninteresting events on a certain port
  - An action is a run-time effect
Abstract Debugging Model

- A debugger observes “primitive events”, and
  - clusters, filters and reports the events
  - influences the run-time (i.e. break and resume)
  - queries the run-time (i.e. current point of execution)
- TIDE event rules guide debugging processes:
  - A port is a cluster of debugging events
  - A condition filters uninteresting events on a certain port
  - An action is a run-time effect
- The collection of ports, conditions and actions is open
  - Theoretically, and in the implementation of TIDE
Event Rules — Ports

- E.g. step, entry, exit, location(\textit{loc}), var-access, send, receive
- Not restricted to a particular programming model
- There is a set of predefined ports
- The “step” port is really needed
Event Rules — Functions in Conditions and Actions

- E.g. cpe, state, var, msg, stack-depth, eq(e1,e2), ...
- E.g. break, resume
- Functions represent information that run-times provide
- There is a set of predefined functions
- “cpe” is really needed, as are “break” and “resume”
Example rules

**Single Stepping Execution**
Port: step, Condition: true, Action: break

**Instruction/Statement highlighting**
Port: stopped, Condition: true, Action: cpe

**Watching variables**
Port: stopped, Condition: true, Action: var(variable name)

**Breakpoints**
Port: location(loc), Condition: true, Action: break
Current Point of Execution

- How to define “step” and ”cpe”?
- “Source-logical” points in the run-time to stop
- Need references to source locations
- Granularity
  - Can be fixed, e.g. like the 'Statement' granularity in C
  - Can be dynamic, e.g. different modes influenced by the user
  - The even rule mechanism would have different ports
Architecture

- Java VM
  - ProcessView
  - Manager
  - DebugProcess
  - ToolBus adapter

- ToolBus server

- (DS)L runtime environment

- TIDE Debugging Protocol
  - P
  - A

- Event rules
- Toolkit
- Implementation effort for developing a TIDE adapter
Communication between tools and adapters

- Event Rules define contracts between tools and adapters
- Tools dynamically publish rules to adapters
- Adapters implement rules, or send 'not implemented' event
- Adapters publish non-boolean action values to the bus
ToolBus

- Middleware bus, socket connections to tools
ToolBus

- Middleware bus, socket connections to tools
- Coordination protocol expressed in ACP-based language
  - Algebra of Communicating Processes
ToolBus

- Middleware bus, socket connections to tools
- Coordination protocol expressed in ACP-based language
  - Algebra of Communicating Processes
- Data encoded as “ATerms” (like XML/Lisp expressions)
Middleware bus, socket connections to tools

Coordination protocol expressed in ACP-based language
  - Algebra of Communicating Processes

Data encoded as “ATerms” (like XML/Lisp expressions)

Adapters for a collection of languages
Implementation effort for developing a TIDE adapter

- Interpreter or runtime is
  - C-based: Reuse toolbus adapter and tide library, implement adapter
  - Java-based: Reuse toolbus adapter and tide library, implement adapter
  - ASF+SDF based: “Everything” is generated
  - Something else: write your own toolbus & tide adapters
Implementation effort for developing a TIDE adapter

- Interpreter or runtime is
  - C-based: Reuse toolbus adapter and tide library, implement adapter
  - Java-based: Reuse toolbus adapter and tide library, implement adapter
  - ASF+SDF based: “Everything” is generated
  - Something else: write your own toolbus & tide adapters

- Growing features incrementally
  - Start with port:step, actions:cpe,break,resume
  - Continue with: stack-depth, var-value, ...
  - Develop specific tools later
Experience — Summary

- Wrappers around existing debugging interfaces
  - gdb, jdb, ToolBus
  - all written in Java, console-based I/O
Experience — Summary

- Wrappers around existing debugging interfaces
  - gdb, jdb, ToolBus
  - all written in Java, console-based I/O
- Domain specific languages
  - ASF — Algebraic Specification Formalism (conditional term rewriting)
  - RScript — Relational Calculus
  - Pico — Little educational imperative language
Evaluation

- Scope: excellent; very different paradigms easily integrated
- Speed: simple/naive adapter implementations cause bottlenecks
  - Use mappings to native breakpoint mechanisms
- Scalability: ToolBus communication bottleneck
  - Scales to 100[0]'s of tools, but no more
  - But! new Java interpreter eats up Java tools
  - Every event, every rule passes through the bus
  - But! new tool communication protocol “cuts out the middleware”
Current work

- Done: ToolBus as an Eclipse service/plugin
- Being done: Reimplementation of Meta-Environment GUI as Eclipse plugins
- Future: Recasting of TIDE “views” to standard Eclipse debugging views??
Nested debugging

- Three languages, three debuggers:
  - C — imperative programming language
  - ASF — term rewriter, implemented in C
  - Pico — imperative language, expressed in ASF

- Nested debugging:
  - Pico interpreter runs a Pico program
  - ASF is running the Pico interpreter
  - C run-time is running ASF

- Showing ASF and Pico levels
Conclusions

- TIDE — Toolbus Integrated Development Environment
  - Abstract debugging protocol based on Event Rules
  - Language & programming paradigm independent
  - ToolBus/Java/Swing implementation
Questions

Phd Thesis
“A Framework for Debugging Heterogeneous Applications”, Pieter Olivier, Universiteit van Amsterdam, 2000

URL
http://www.meta-environment.org

Future
Suggestions, references, ideas, programmers welcome!