Target Communication Framework

Monitoring/Tracing Workshop, Jan 29-30, 2008

Felix Burton (felix.burton@windriver.com)

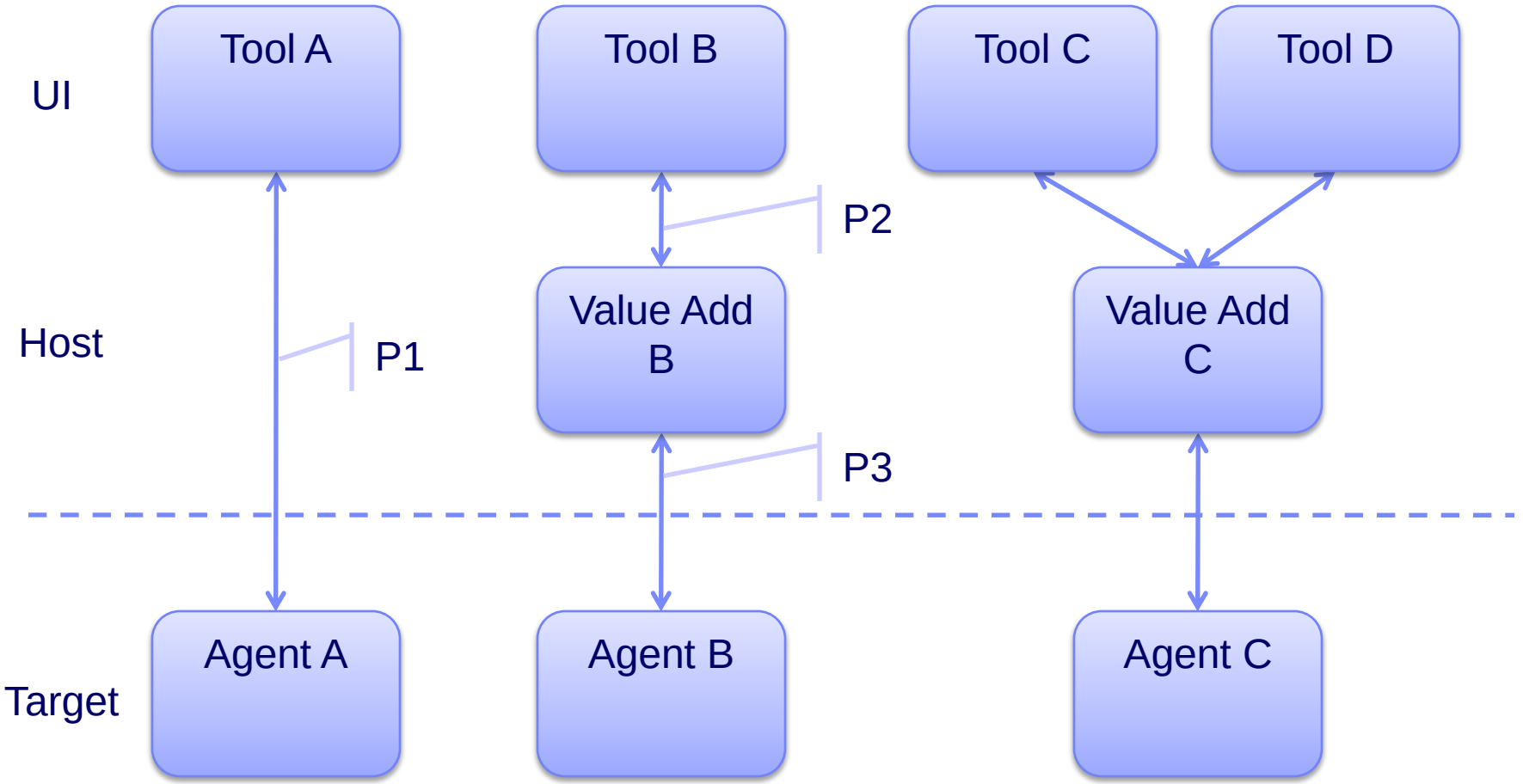
Agenda

- Out Line
- Motivation
- Core Design Ideas
- Example use cases
- Monitoring/Tracing Challenges
- Status

Out Line

- Define an open end to end tool to target communication mechanism for the purpose of development, debug, monitor, analysis and test
- Specification
  - Transport channel supporting extensible set of “services”
    - Typically on top of a TCP/IP stream, but other transports supported as needed but the target
  - Services defining commands, progress, replies, events & semantics
  - Discovery of available servers and services
- Prototype implementation
  - Eclipse plug-ins
  - C-based agent
- Scope
  - Cross tools (i.e. host and target are different) benefits the most, but is applicable to native tools as well
  - Target agent, OCD/JTAG and simulator connections

Existing Architectures



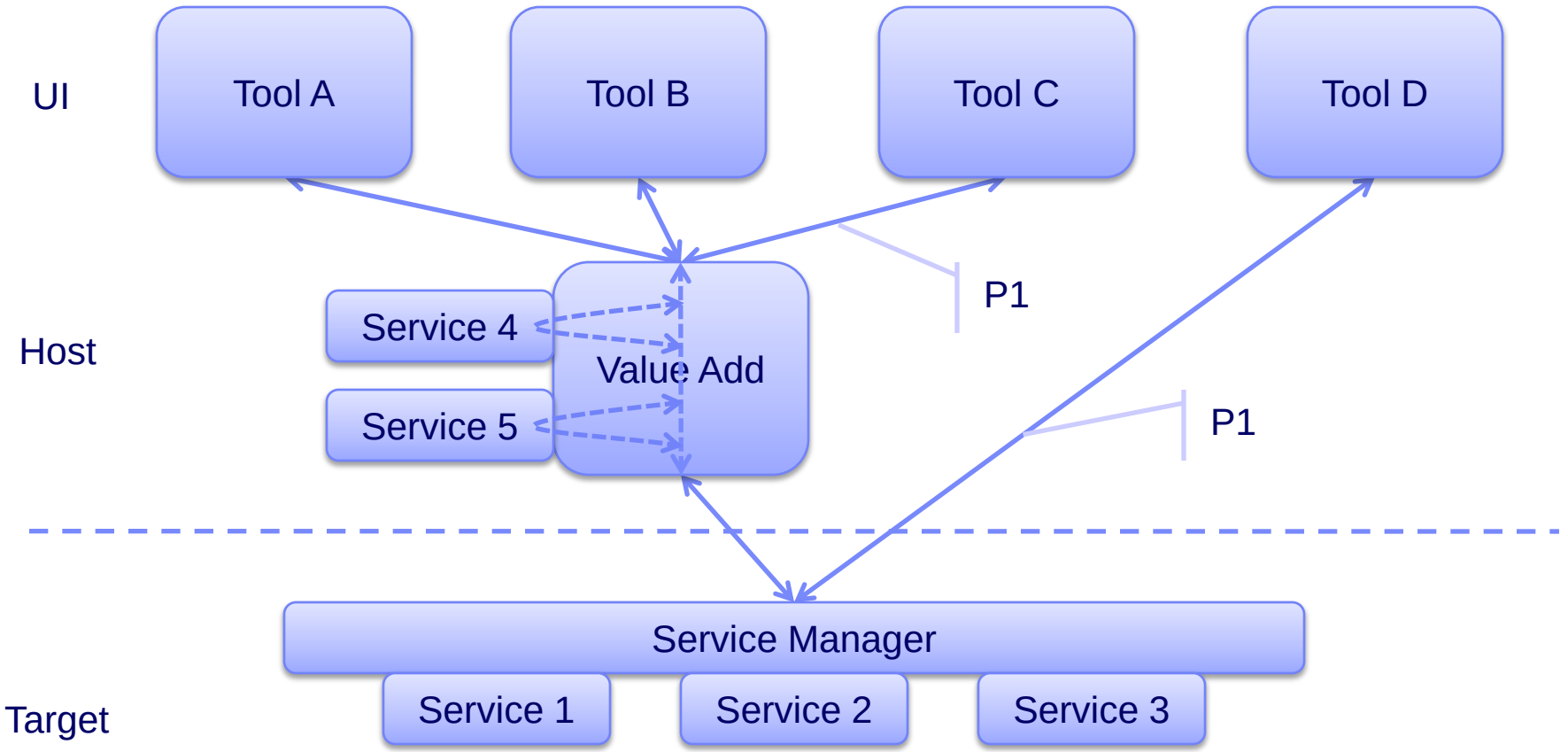
Motivation

- Almost every cross development tool have their own infrastructure (agent, connection, protocol, setup, etc)
- This leads to:
  - Poor user experience
    - Each tools has its own target configuration
    - Increased target intrusion (footprint, multiple agent interaction)
    - Inconsistent product availability matrix
  - No sharing between agents
    - Duplicated maintenance effort
    - New features have to be added in multiple places
    - New tools have to start from ground zero
  - Limited Eco-system

Core Design Ideas

- Service knows best how to represent the system – get information from there and data-drive layers above
  - If not possible, put the knowledge in the lowest possible layer and data drive the layers above
- Use the same protocol end-to-end, but allow value-adding servers to intercept select services when needed and pass-through everything else
- Services as building blocks that can be used by multiple clients (tools) for different environments (target agent, OCD, simulator)
  - Avoid tools specific agents
  - Bridge gap with environment specific services to setup/configure common services
- Support high latency communication links

Architecture Overview



Use Case: SimpleJtagDevice

- Debug (run-control, breakpoint, memory register)
- Possibly Others (flash programming, download, etc)

Use Case: TestExceutionAgent

- Process launch and kill
- Standard I/O redirection
- File system access

Use Case: LinuxUserModeAgent

- Debug (run-control, breakpoint, memory, register)
- OS Awareness (process/thread list, CPU utilization, etc)
- Process launch and kill
- Standard I/O redirection
- File system access
- Monitoring (event-config, event-log)

Performance/Monitoring Challenges

- Multiple tools sharing instrumentation points and HW counters
- Common event format
- Specifying flexible triggers and filters
- Scalability for large event logs

Prototype Status

- Eclipse plug-ins
  - Remote file system access and simple top style monitoring integrated with RSE
  - Basic debugging integrated with Eclipse Core Debug plug-ins
  - DSF integration in the works
- C-based agent
  - Services needed by Eclipse plug-ins
  - Simple UDP based discovery
  - Functional on Linux and VxWorks

Specification Status

- Transport Channel
- Current Services
  - Run Control, Memory, Register, Breakpoint, Processes, Stack Trace, File System, System Monitoring
- Review of current and specification of additional services in power.org and Eclipse

Links

- Prototype source repository
  - svn://dev.eclipse.org/svnroot/dsdp/org.eclipse.tm.tcf/trunk
  - http://dev.eclipse.org/viewsvn/index.cgi/org.eclipse.tm.tcf/?root=DSDP_SVN
- FAQ
  - http://wiki.eclipse.org/DSDP/TM/TCF_FAQ

QUESTIONS/COMMENTS?