

Tutorial #22

Using Test and Performance Tools Platform (TPTP) Logging and Monitoring Tools

Antony Miguel
Scapa Technologies

Paul Slauenwhite
IBM Rational Software

Agenda

- Introductions
 - Speakers
 - Participants [Optional]

- What is Problem Determination?
 - Problem Statements
 - Taxonomy
 - Tooling
 - Benefits

- Eclipse

- TPTP Project
 - Mandate
 - Participants
 - Composition

- Eclipse Modeling Framework

Agenda (continued)

- EMF Models
 - Trace Events
 - Common Base Events
 - Statistical Events

- Agent Controller
 - Agent Architecture
 - Trace Agents
 - Logging Agents
 - Statistical Agents

- Common Base Event
 - Design

- Generic Log Adapter
 - Architecture
 - Adapters

Agenda (continued)

- Logging Scenarios
 - Architecture
 - Logging Facility Support
- Log and Trace Analyzer
 - Architecture
 - Importing and Monitoring
 - Profiling and Logging Perspective
 - Symptom Databases
 - Analysis
 - Correlation
- Break
- Demonstrations
 - Statistical Monitoring and Analysis
 - Logging

Agenda (continued)

- Conclusion
 - Project Extensibility
 - Future work
 - Resources
- Questions, Answers and Discussion
- Appendix 1- Java Logging XML Log File
- Appendix 2- logger.dtd

Introductions - Speakers

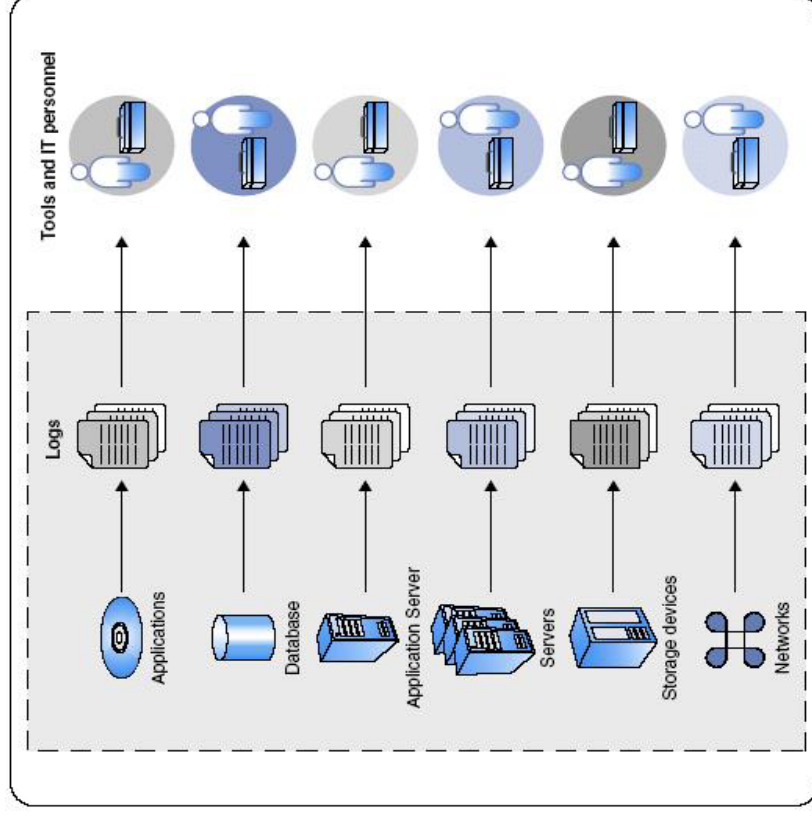
- Antony Miguel
 - Scapa Technologies
 - antony.miguel@scapatech.com
- Paul Slauenwhite
 - IBM Rational Software
 - paules@ca.ibm.com

Introductions – Participants [Optional]

- Name and city.
- Company or academic institution.
- Eclipse and TPTP experience.
- Tutorial objectives.
- Any other interesting information?

What is Problem Determination? - Problem Statements

- Modern computing systems are becoming more complex with the rapid development of raw computing power (e.g. processing speed and storage capabilities) and distributed technologies.
- Computing systems are more fragmented with the proliferation of distributed technologies, such as the Internet.
- Numerous heterogeneous products and applications are now required to build modern computing systems.
- Pervasive and ubiquitous computing devices pose new management complexities.
- Shortage of highly skilled workers to manage and maintain complex heterogeneous computing systems.
- Problems are growing exponentially due to an equally increasing use of complex computing technology.



What is Problem Determination? - Taxonomy

- Monitor, evaluate, detect, analyze, correlate, resolve and diagnose the root cause of run-time problems in complex end-to-end heterogeneous computing systems.
- Most computing systems generate log and/or trace data which is a rich source of system activity and control flow information.
- Useful for detecting and resolving run-time problems such as configuration errors, performance degradation, exception states, resource starvation, security failures, communication delays and deadlocking.
- However:
 - Typically persisted in large (MB – GB) files.
 - Standards and terminology formats vary between vendor, product and version.
 - Varying levels of detail (e.g. terse and obscure).
 - Distributed systems scatter files across the network.
 - Textual viewing and manual searching or proprietary tooling.
 - No detailed explanation or solutions for records.
 - No relationship between records and files.

What is Problem Determination? - Tooling

- Centralized tooling to collect and consolidate log and trace data from disparate systems into a single management tool.
- Based on open standards to be useful and to promote widespread adoption in a heterogeneous environment.
- Promotes a common event format for vendor, product and version-specific log and trace formats.
- Consumes a common event format for viewing, navigating, sorting, filtering and searching large amounts of log and trace data.
- Correlates log and trace records to determine sets of related events thereby visualizing control flow across distributed systems.
- Analyzes log and trace records to provide explanation and possible solutions to known problems.

What is Problem Determination? - Benefits

- Extensible open-source tools and technology with consistent and public interfaces and points of control for industry-wide acceptance, agreement and adoption.
- Assist end-users, administrators, field service engineers and developers in decreasing the time required for debugging, maintaining and securing complex computing systems.
- Increase return on administration investment by reducing maintenance costs and improving asset utilization.
- Rapid problem determination translates into increased system stability, security, availability, resiliency and overall QoS.
- More efficient systems to better focus resources on business processes.
- Problem Determination = ↓ Costs + ↑ Stability

Eclipse

- An open-source project dedicated to provide an universal, robust, full-featured and commercial-quality platform written in Java for developing Integrated Development Environments (IDEs).
- Founded by IBM and other industry leaders in November 2001.
- Composed of projects, each of which is overseen by a Project Management Committee (PMC) and governed by its Project Charter.
- Each project is composed of its own subprojects and is licensed under the Eclipse Public License (EPL) starting in 2005.

TPTP Project - Mandate

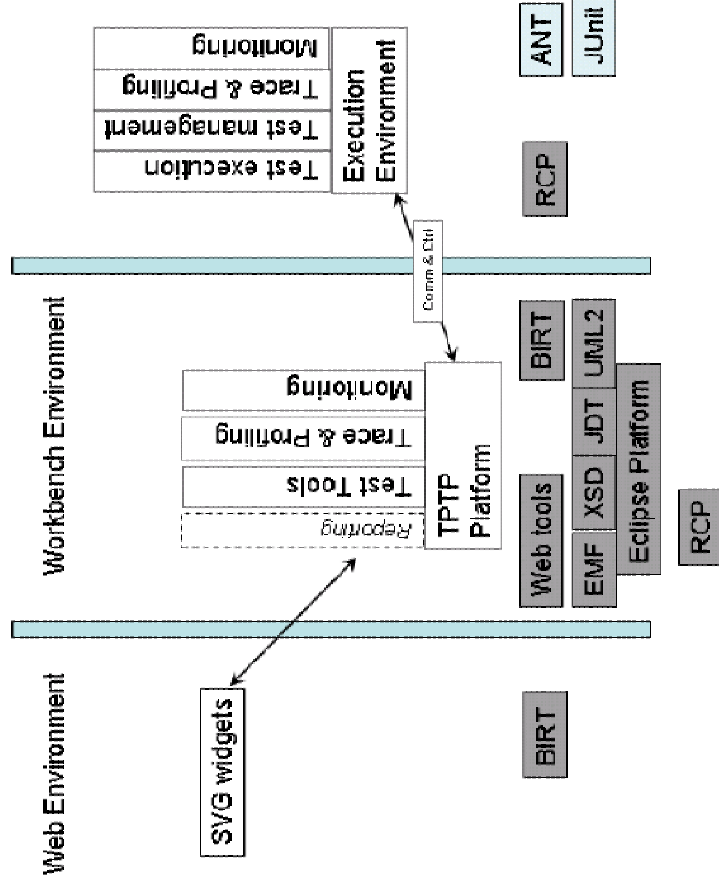
- Started in 2002 as the Eclipse subproject named Hyades.
- Promoted in 2005 to an Eclipse project and renamed as the Test and Performance Tools Platform (TPTP) project.
- Open-source platform for Automated Software Quality (ASQ) tools including reference implementations for testing, tracing and monitoring software systems.
- Extensible framework and infrastructure that embraces automated testing, trace, profiling, monitoring, and asset management.
- Includes an unified data model, normative user experience and workflow and an unified set of APIs and reference tools that work consistently across a range of targets.
- Principles:
 - Extension of the Eclipse Value Proposition
 - Vendor Ecosystem
 - Vendor Neutrality
 - Standards-Based Innovation
 - Agile Development
 - Inclusiveness & Diversity

TPTP Project - Participants

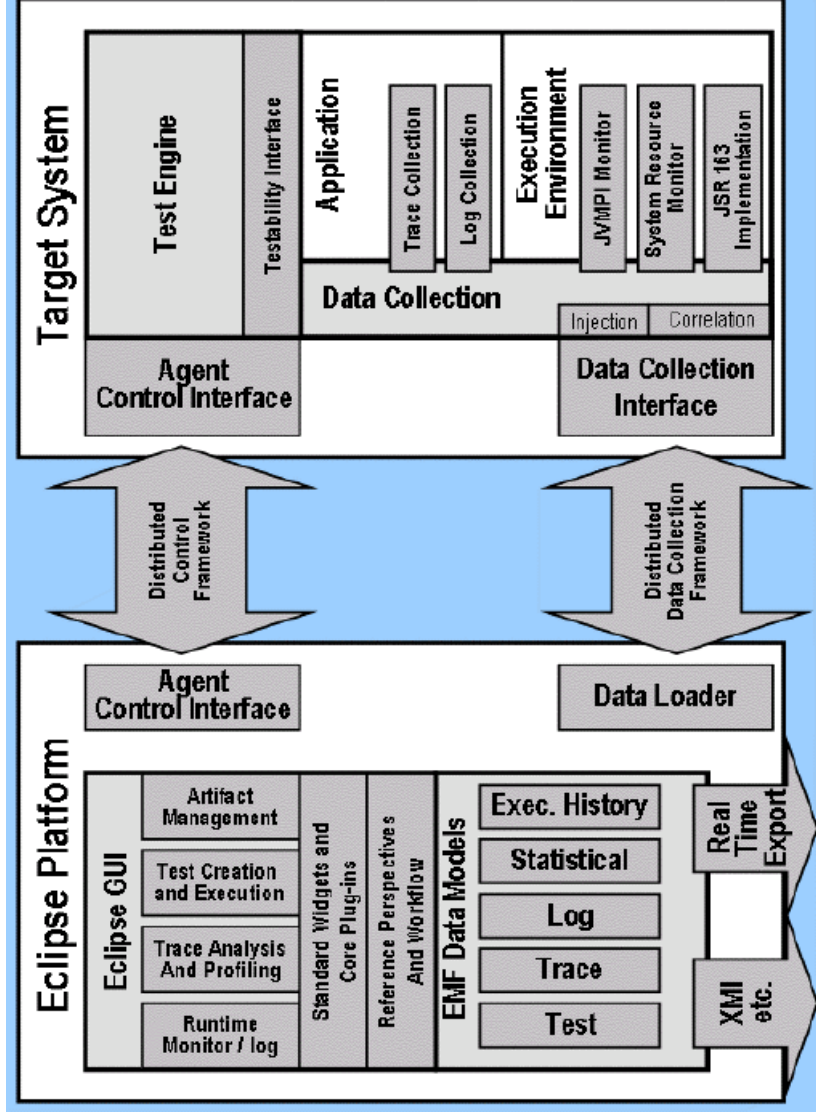
- Computer Associates (Test and monitoring design)
- Compuware (Monitoring Design)
- FOKUS (Test design)
- IBM (Trace, Test, Log, Data Collection)
- Intel (Data collection)
- OC Systems (BCI - Data Collection, Probe Kit)
- SAP (Test design)
- Scapa Technologies (Test, Data Collection)

TPTP Project - Composition

- TPTP is made up of 4 Projects:
 - Platform – core TPTP functionality including all data models.
 - Test – test UI and execution.
 - Trace – tracing and probes.
 - Monitoring – monitoring and logging.
- Each TPTP Project is made up of Components:
 - E.g. UI, agents, execution etc..
- Organized around a client or consumer and server or producer model:
 - May be on the same or different physical machines.



TPTP Project - Composition



Eclipse Modeling Framework (EMF)

- Describe data objects, attributes, relationships etc..
- Create models as Class Diagram or XSD and generate to Java code.
- Use generated Java APIs to create, manipulate and persist models.
- Models persisted as XMI files.
- Resource persistence layer abstraction allows use of a relational database as the file system.
- Easy creation of data models without the headaches of persistence and serialisation.

EMF Models

- TPTP EMF Hierarchy Models
 - Trace
 - Log (Common Base Event)
 - Statistical

- Other TPTP EMF Models
 - Test
 - Behavioural

- All hierarchy models contained within a single TRCAgent (*.trcxmi) model, representing a trace/data collection session.

- Models are populated via standard XML fragments parsed by standard model loaders.

EMF Models – Trace Model

- Contains application and/or system trace information (e.g. stack trace information).
- Two main areas to the trace model:
 - Type definition and statistical snapshot structure.
 - Stack tracing structure.
- Types (e.g. classes & methods) are defined once in type definition branch.
- Statistical snapshot info (e.g. # of times method called) can be stored in type definitions.
- Stack tracing structure contains any stack traces and references type definitions from earlier part of model (e.g. method X in class Y called by thread Z).

EMF Models – Common Base Event

- Contains logging events and messages.
- Basically a list of Common Base Event objects.
- Each Common Base Event has a wide range of possible attributes.
- Common attributes include:
 - Date / Time.
 - Message (string text).
 - Source component / subcomponent.
 - Severity (info, warning, minor error, fatal error ...).
 - Sequence Number (if time ordering is not accurate).
 - Repeat Count (e.g. same event occurred 5 times).

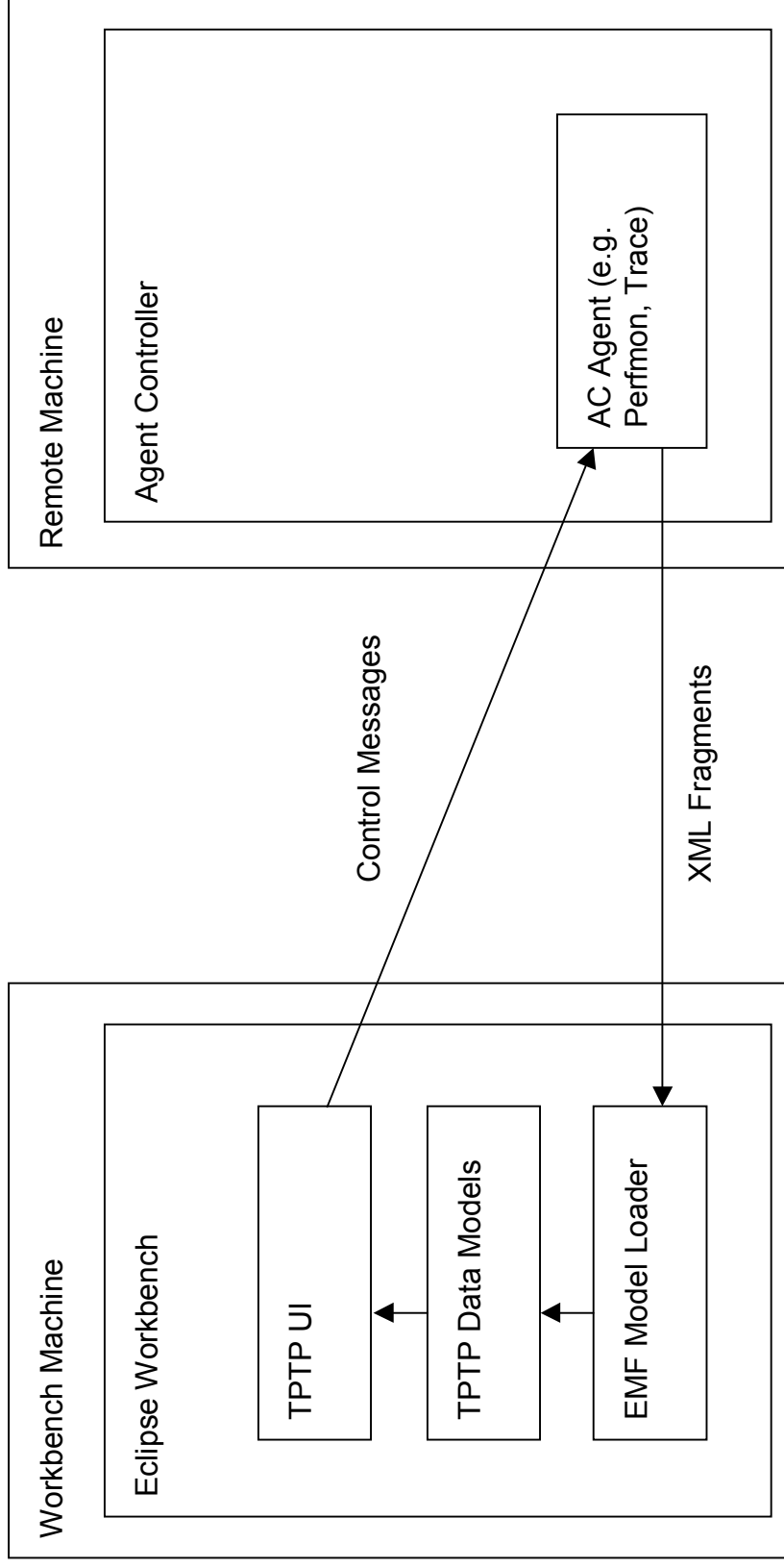
EMF Models – Statistical Model

- Contains general statistical data.
- Descriptors make up an arbitrary hierarchical structure.
- Counter Descriptors are parts of the structure that may contain data (may contain observations).
- Snapshot Observations contain statistical data:
 - Discrete Observation contains integer / time pairs.
 - Contiguous Observation contains float / time pairs.
 - String Observation contains String / time pairs.

Agent Controller

- Standalone component, not necessarily installed with the TPTP workbench.
- Flexible daemon used to bridge gap between client and SUT (system under test/trace).
- Has no native data collection or control capacity, all SUT monitoring functionality comes from agent extending the AC.
- One simple agent written to gather SUT data and transfer it back via the AC means that agent can now be deployed to monitor any part of the SUT.
- A standard XML loader at the client side receives and parses the XML fragments passed back from any agent.
- XML Loader populates standard TPTP data models which can then be viewed and manipulated in the standard TPTP UI tools.

Agent Controller – Agent Architecture



Agent Controller – Trace agents

- Available trace agents in v3.1:
 - Java JVMLI agent:
 - Uses same JVM APIs as standard HPROF agent but sends traced information to a TPTP trace model for analysis.

Agent Controller – Logging Agents

- XML message-based agents for real-time monitoring of message generating facilities (e.g. loggers).
- Extensible architecture for crafting proprietary Logging Agents:
 - Defined APIs and programming model.
- TPTP provides Logging Agent support for several popular Java-based logging and tracing facilities.

Agent Controller – Statistical Agents

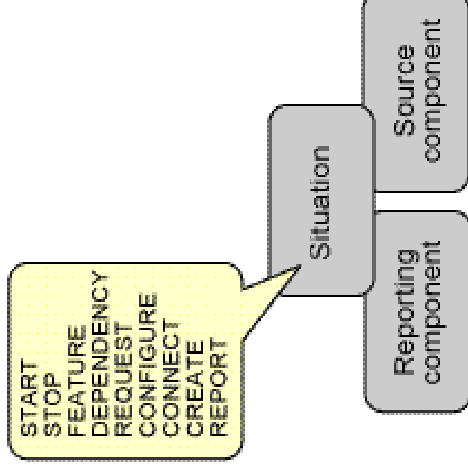
- Available statistical agents in 3.1:
 - Windows Host (Perfmon):
 - Standard Perfmon API metrics – CPU, network, disk etc..
 - Linux Host:
 - Standard /proc metrics – CPU, network, disk etc..
 - JBoss J2EE Application Server:
 - Counters exported by JBoss server (not by the Application running on the server).
 - JOnAS J2EE Application Server:
 - Similar to JBoss agent.

Common Base Event

- Open-source specification (OASIS) to provide a *common* and standardized taxonomy for *events* occurring in hardware and software.
- Unified format and terminology for the standardized exchange and consistent interpretation of problem determination data to circumvent varying vendor, product and version representations.
- Provides a mutual format for event data enabling the interoperability of problem determination technologies and tooling thereby decreasing the costs of managing complex computing systems.
- Designed to abstractly describe all events within a multi-component system but sufficiently granular to describe any occurrence or incident.
- Common Base Event XML schema defines the overall structure of the event, format of each property and all mandatory properties for completeness.
- TPTP provides EMF producer (e.g. native logging) and consumer (e.g. model artifacts) Java implementations.

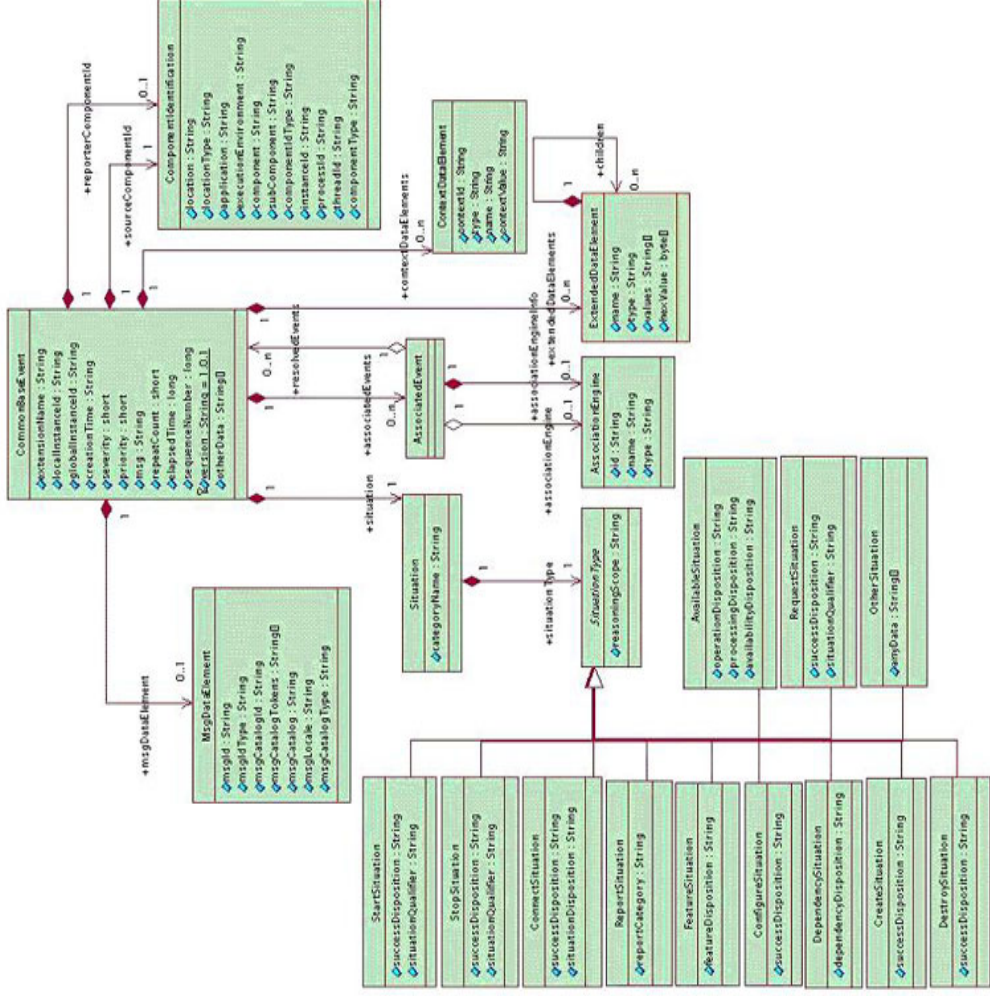
Common Base Event - Design

- Events consist of message(s) and metadata resulting from an occurrence or situation including autonomous, logging, tracing, management and business situations.
- Events are derived from situations that are based on a structured *3-tuple* format:
 - Situation data, the properties describing the situation including correlation information.
 - Component impacted by a situation, or the source
 - Component observing a situation
- All other required and optional event properties provide context for the situation.



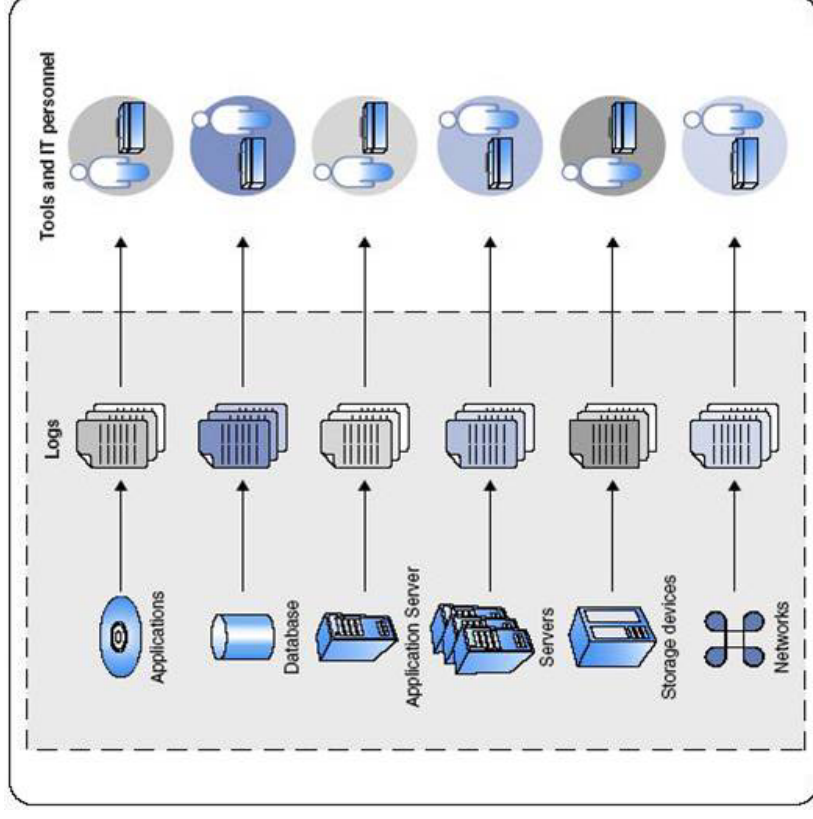
- **Situation** defines a change in state that the component is programmed to inform
- **Reporting component** is the component reporting the problem
- **Source component** is the component experiencing the problem

Common Base Event - Design



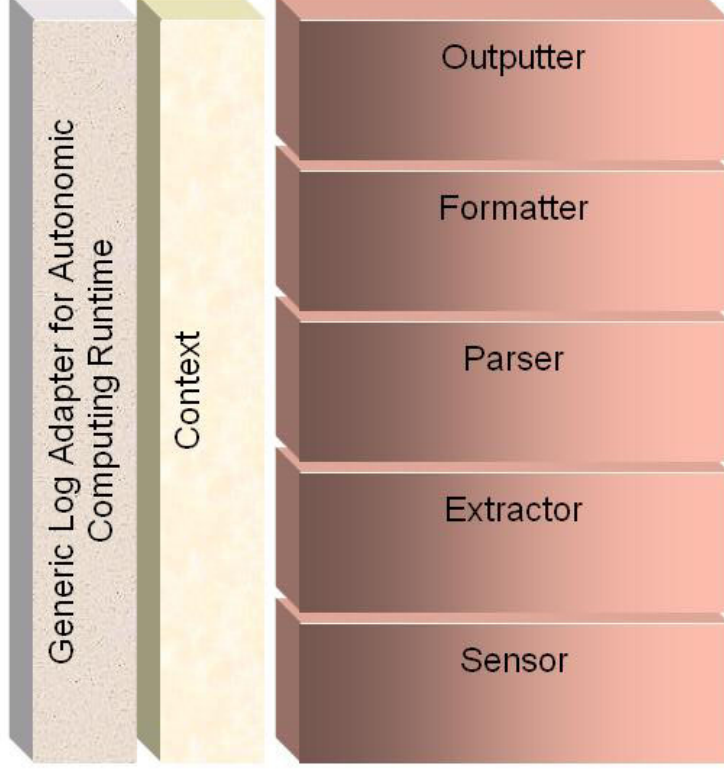
Generic Log Adapter

- Tooling that transforms log and trace data in proprietary formats into the Common Base Event format for use as a problem determination resource.
- Assists developers in adapting log and trace data to the Common Base Event format without re-writing existing applications.
- Originated from IBM Research as a mechanism to cheaply enable Common Base Event compliance in legacy computing systems.



Generic Log Adapter - Architecture

- Based on a pluggable pipe and filter architecture.
- Filters or components are organized by contexts which are associated with one specific data set.
- Scalable to support large amounts (MB – GB) of log and trace data.
- Four modes of operation:
 - Batch or continuous.
 - Static or rules.



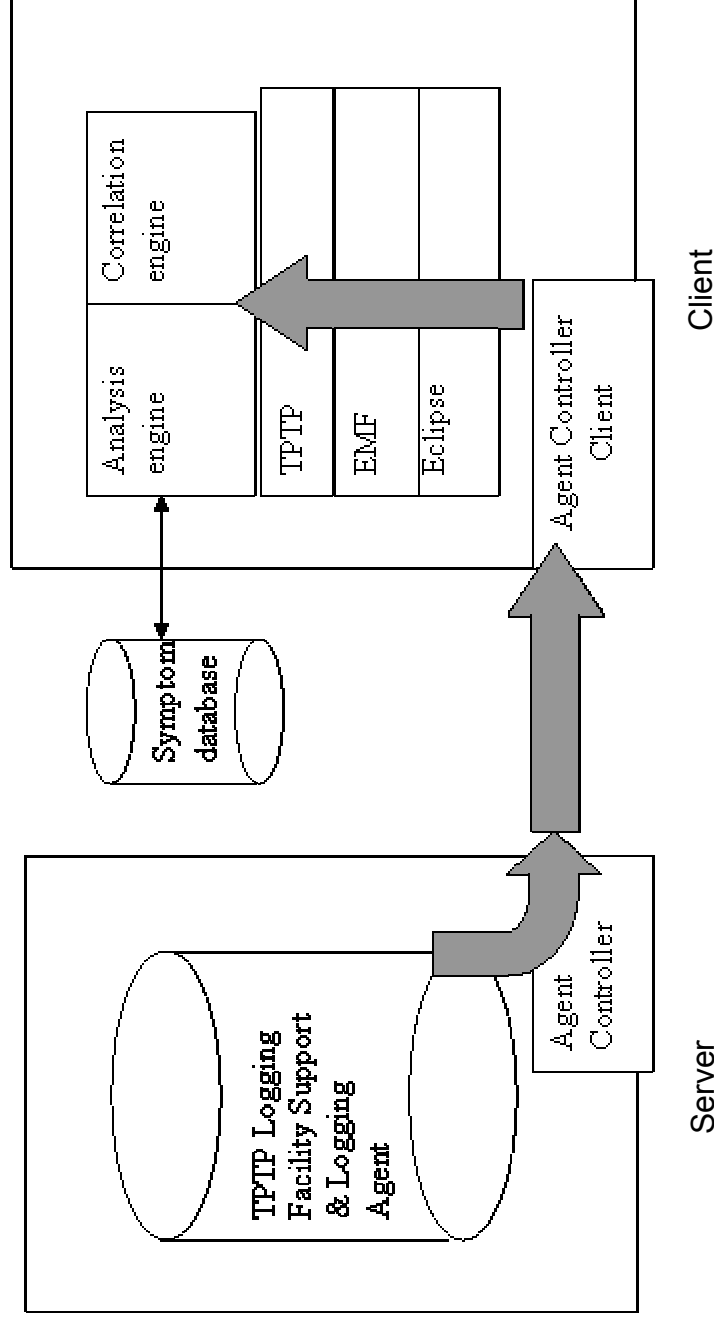
Generic Log Adapter – Adapters

- Transforms are defined in adapter files:
 - Contains one or more contexts and the components for each context.
 - Stored as XML based on a defined schema.
- Parsing component incorporates mapping proprietary log and trace record properties to Common Base Event properties using mapping rules.
 - Rules are defined using static strings or regular expressions (e.g. Perl).
 - Static Java parsers may be used to parse data.
- Adapter Configuration Editor provides a tool to write, debug and test adapter files using a template log file.
- Adapter files may be mechanically converted into Log File parsers for use in the Log and Trace Analyzer.

Logging Scenarios

- New or redesigned systems may be instrumented to natively generate Common Base Events using the producer Java implementation.
- However:
 - Java-based logging and tracing facilities often enforce proprietary record formats.
 - No existing integration with tooling for real-time monitoring of loggers and logged records.
- Hyades provides the following support for several popular Java-based logging and tracing facilities:
 - Common Base Event logging.
 - Logging Agent sinks.
 - Filtering.
 - Configuration.
 - Formatting.

Logging Scenarios - Architecture



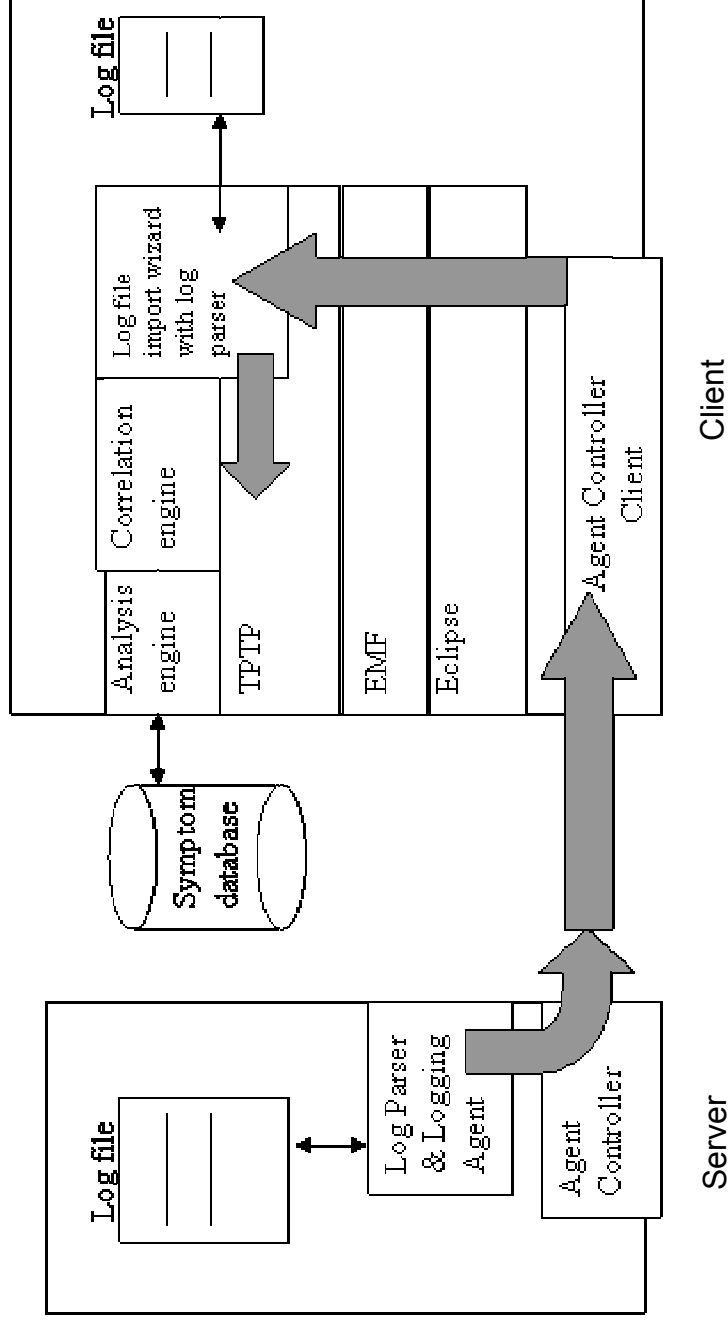
Logging Scenarios - Logging Facility Support

- TPTP provides Common Base Event and Logging Agent standalone and plug-in support for the following popular logging facilities:
 - Jakarta Apache Commons
 - Java Logging (JSR-047)
 - Jakarta Apache Log4J
- Extensible architecture for crafting proprietary Logging Agents:
 - Defined APIs and programming model.

Log and Trace Analyzer

- Eclipse-based tooling for collecting, persisting, viewing, navigating, sorting, filtering and searching EMF model data.
- Permits users to monitor, evaluate, detect, analyze, correlate, resolve and diagnose the root cause of run-time problems in complex end-to-end heterogeneous computing systems.
- Extensible architecture to allow vendor and product specific EMF model views and problem determination tasks.

Log and Trace Analyzer - Architecture

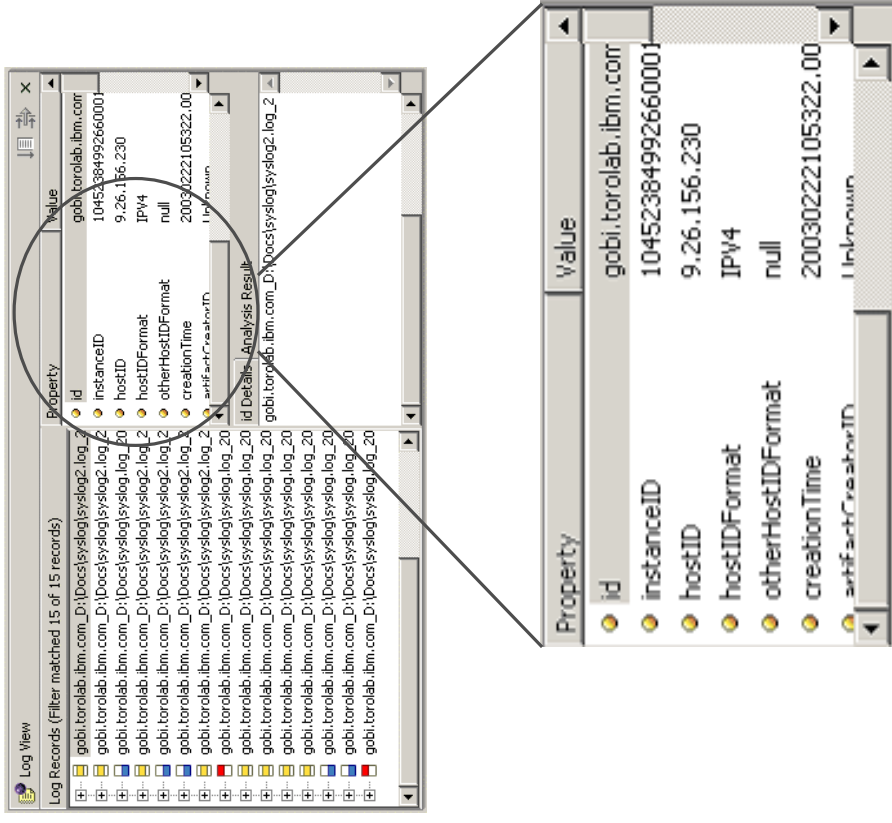


Log and Trace Analyzer - Importing and Monitoring

- Tooling for batch importing of log and trace files into the Common Base Event EMF models using Generic Log Adaptor static and rules parsers.
- Log and trace files may be imported from the local and/or remote machine.
- Real-time monitoring of Logging Agents to populate Common Base Event EMF models.
 - Jakarta Apache Commons
 - Java Logging (JSR-047)
 - Jakarta Apache Log4J
 - Proprietary Logging Agents
- Extensible architecture to allow vendor and product specific log and trace file parser and Logging Agents.

Log and Trace Analyzer - Profiling and Logging Perspective

- Set of views to display log, trace, statistical and profiling EMF model data in a well organized format.
- Hierarchically organized by folder, monitor, host, process and varying agents.
- Log view provides a nested tree-like display for viewing, navigating, sorting, filtering and searching for large amounts of log and trace data.
- UML Sequence Diagram view provides a graphical representation of interactions.
- Statistical console provides a visual display with graphing capabilities for the statistical EMF model.
- Extensible architecture to allow users to define vendor and product specific views.



Log and Trace Analyzer – Symptom Databases

- A *database* consisting of matching patterns indicating known problems, explanations and resolution steps.
- Persisted as an XML file with a defined schema.
- Crafted using the Symptom Database Editor in the Log and Trace Analyzer.
- Vendors and organizations may provide multiple hierarchical symptom databases based on logical or business divisions.
- Local and remote (FTP/HTTP) importing functionality to ensure content freshness.

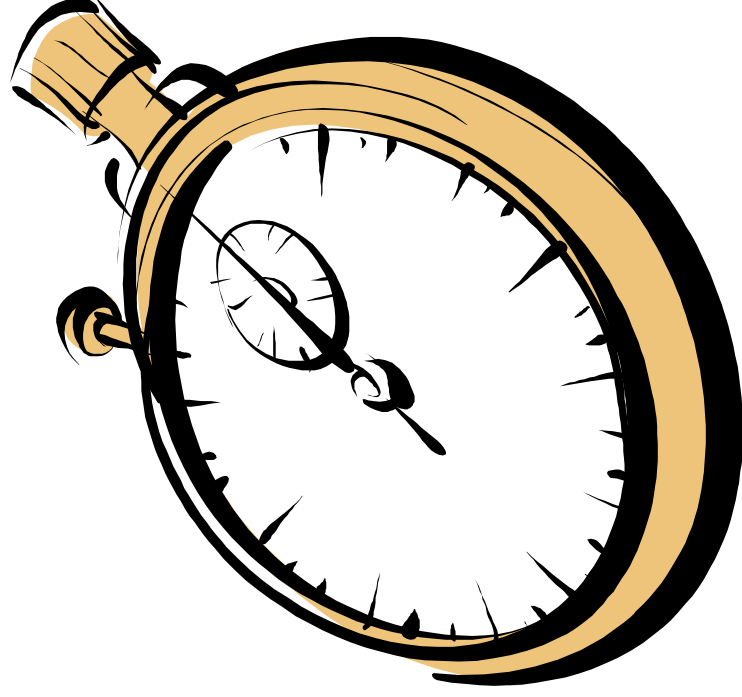
Log and Trace Analyzer - Analysis

- Allows users to easily detect and solve problems that have already been previously encountered.
- Analysis consists of lexicographically comparing varying Common Base Event properties with match patterns in one or more symptom databases.
- Extensible architecture to allow users to define vendor and product specific analysis engines.

Log and Trace Analyzer - Correlation

- Computing system maintainers require a detailed understanding of an entire computing system in order to detect and resolve cascading problems.
- Correlation determines one or more sets of related events to visualize control flow within and between computing systems.
- A correlation engine or schema associates varying Common Base Event properties based on a predetermined criteria (e.g. time).
- Extensible architecture to allow users to define vendor and product specific correlation engines or schemas.

Break (15 minutes)



Demonstrations

- Attendees are not required to have a computer.
- Configuration for attendees with computers:
 - Hyades v3.2.0 Release Build:
 - Hyades Runtime
 - Hyades Examples
 - Hyades Data Collection Engine (platform dependant)
 - <http://www.eclipse.org/hyades/> >> Download Page >> Release Builds >> 3.2.0
 - See the Requirements section for the Hyades dependencies.
 - See the Related Documents section for the release notes and installation guide.

Demonstrations – Statistical Monitoring and Analysis

- Launch Configurations and Monitoring
 - Profiling and Logging Perspective: Profiling Monitor view
 - TPTP Data Model hierarchy and Agents
 - Launch Configurations
 - Statistical Data view

- Statistical Comparison and Analysis
 - Statistical Console editor

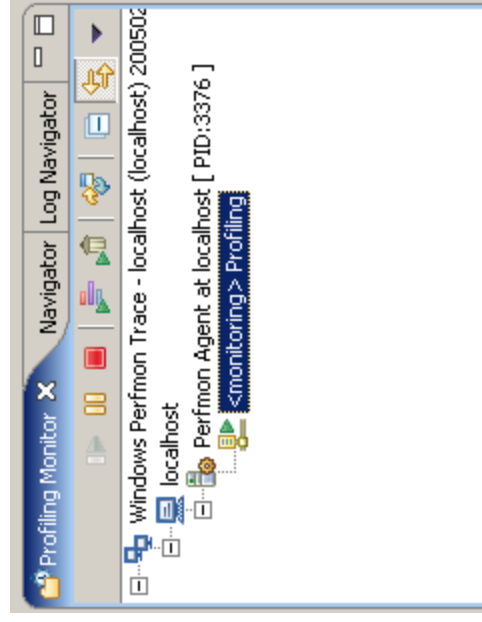
Launch Configurations and Monitoring

- Profiling and Logging Perspective
 - Window -> Open Perspective -> Other -> Profiling and Logging
- Profiling Launch Configurations
 - Run -> Profile...
 - Create a new “Host – Windows (Perfmon)” launch configuration
 - Specify “localhost” as the host to be monitored
 - Click “Profile...”
 - Launch configurations for each type of statistical agent
 - Host – Windows (Perfmon)
 - Host – Linux
 - J2EE App Server – JBoss
 - J2EE App Server - JOnAS

Launch Configurations and Monitoring

- Profiling Monitor view
 - Maps the models hierarchy (Monitor, Node, Process, Agent) to a tree
 - Option to hide and show parts of the hierarchy
 - Option to link with views to automatically update content of view based on selection
 - Sessions commonly grouped by Monitor
 - Context sensitive menu on Agent node provides certain controls

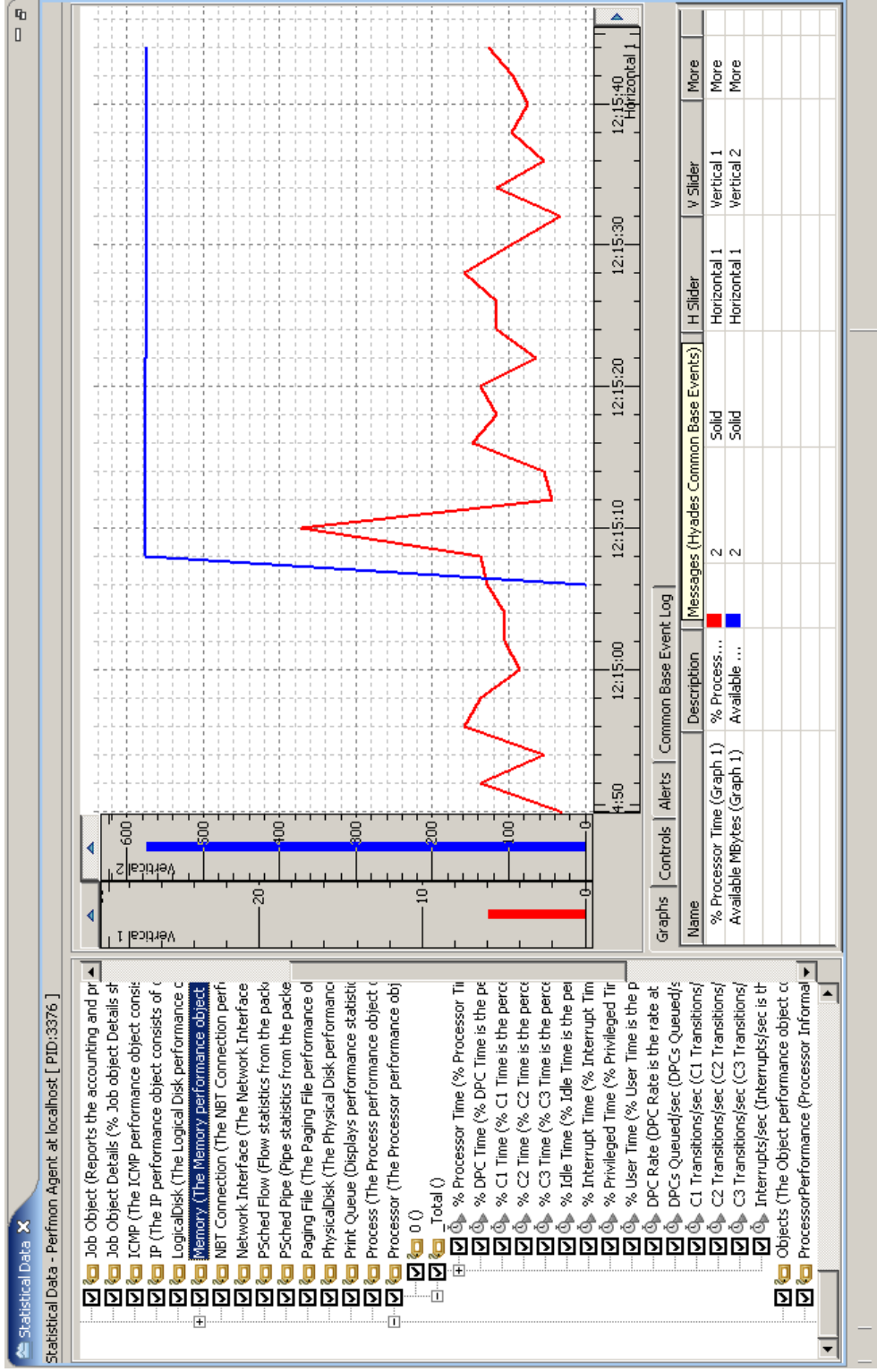
Profiling Monitor view



Launch Configurations and Monitoring

- Statistical Data view
 - Right-click on agent -> Open With -> Statistical Data view
 - Right-click on agent -> Configure Statistical Data Collection Counters
 - Right-click on tree nodes to get children or start monitoring
 - Processor
 - Total
 - % Processor Time
- Statistical Data view provides line graphs of gathered statistical model metrics
 - Easy rescaling to view different sections of data
 - Easy configuration and organisation of metrics to present large swathes of data in a simple way

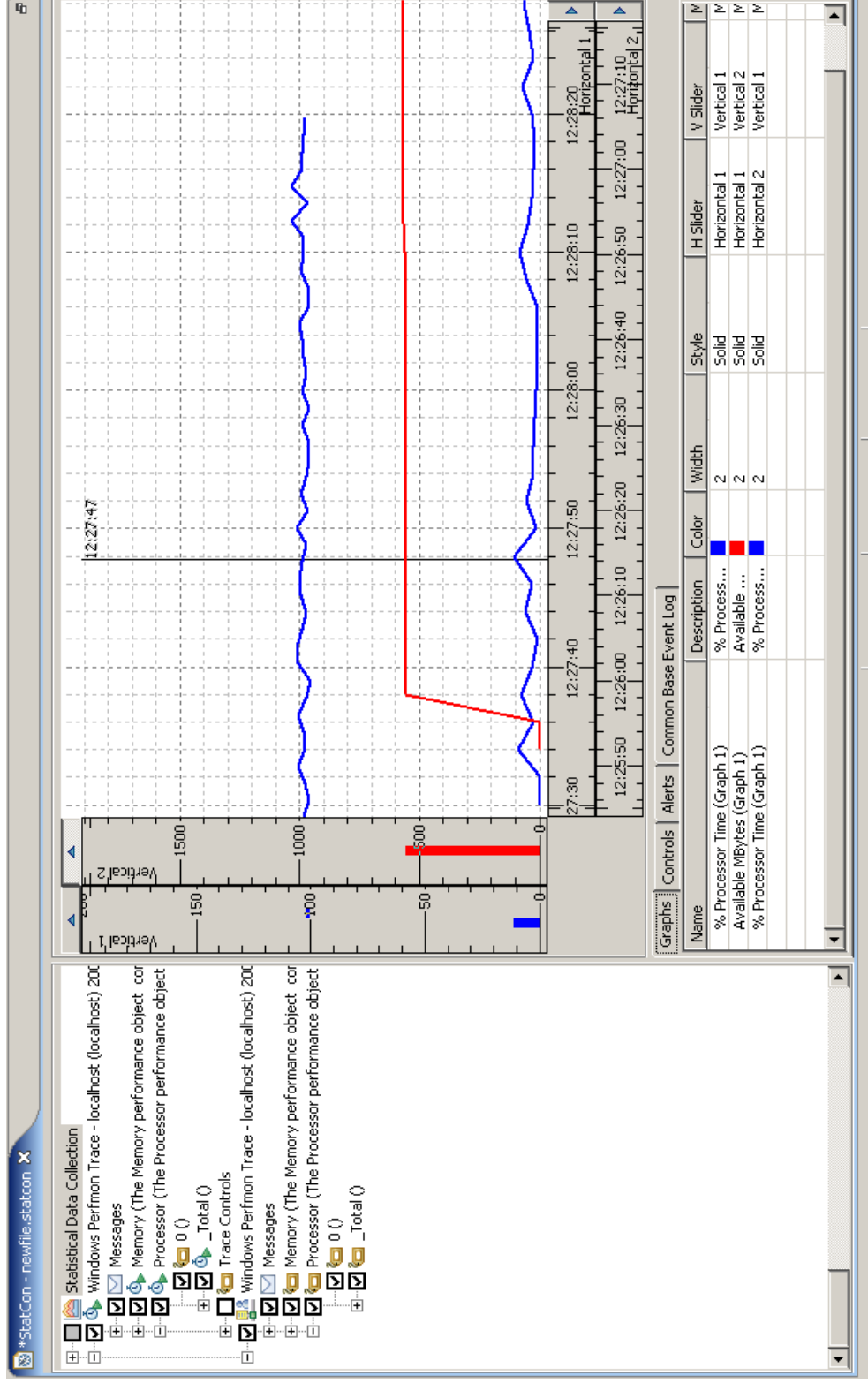
Statistical Data view



Statistical Comparison and Analysis

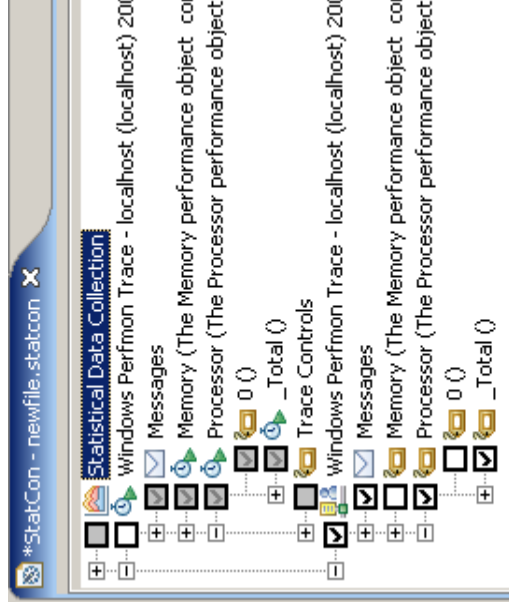
- Statistical Data view useful for short monitoring sessions
- What about saving configurations?
- What about comparing multiple statistical models?
- Answer: Statistical Console editor (.statcon files)
 - Same basic interface as the statistical data view
 - Can add/remove live or historical statistical models
 - Can save and reload entire configuration for more complex analysis over time

Statistical Console editor



Common features – Tree Area

- Checkboxes on tree nodes specify filtering of metrics
 - Allows filtering at any level in the tree
 - Switch on/off a whole trace just by using it's root level checkbox
 - Configure more detailed filtering lower down the tree by switching on/off individual metrics
- Associate whole areas of metrics to particular scales in the Graph area



Common features – Graph area

- Vertical scales allow rescaling and transposing of value axis
- Horizontal scales allow rescaling and transposing of time axis
- Line graph averaging can be static or tied to horizontal scale ticks (averaging period changes according to scale)
- Multiple scales allow easy comparison of metrics
 - E.g. memory in MB on one scale (0 – 1000), processor usage on another scale (0 – 100)
 - E.g. collection session from yesterday on one horizontal scale, collection session from today on second horizontal scale
- Snap to associated graphs feature automatically rescales to show all associated graph data



Common features – Graphs table

- Summarises currently viewed (unfiltered) graphs
- Metric Name and Description taken from Statistical model
- Selection in table highlights metric
 - in Graph area to quickly distinguish it from other lines
 - in Tree area in case of naming conflicts in the model
- Allows configuration of line graph for each metric
 - Line colour / style / thickness
 - Scale associations
 - Line plotting type (average, min, max, std dev, min+max etc)
 - Line plotting period (5 secs, 3 pixels, 1 scale tick)
 - Absent data behaviour (draw nothing, draw previous value, draw 0)
 - Offsets (time offset, vertical offset, multiplier)

Common features – Alerts table

- Allows creation of Alerts over statistical model metrics
- Most useful on live data
 - Receive an email when CPU usage goes over 90%
 - Log to CBE model when free Memory goes below 50MB averaged over 1 minute
- Create Alerts over a statistical model observation (graph node in Tree area)
 - Right-click on graph node -> Add alert over this Observation
- Configure alert type and period similar to graph
 - Average / Min / Max etc. 5 sec, 1 min etc.
- Trigger and reset on upper or lower value
 - E.g. trigger on >90, reset on <80
 - E.g. trigger on <50, reset on >50

Common features – Alerts table

- Alert trigger show in table via warning icon
- Assign one or more actions to Alert trigger and reset
 - Log to file (simple ASCII text log)
 - Send an email (SMTP)
 - Play a sound
 - Log as a common base event to parent Agent of statistical model
 - Gathering CPU in a Windows/Perfmon data collection session
 - Create an Alert over the CPU
 - Receive an email and log to session CBE model when CPU goes over 90%

Data Collection Agent Example (optional)

- Two steps to building your own data collection agent
 - Write the AC agent
 - Write the workbench UI (Launch Configuration)
- Simple Agent example
 - Two plugins
 - “eclipsecon.tutorial.datacollection” (workbench UI)
 - “eclipsecon.tutorial.datacollection.agent” (AC agent)
 - Simple agent
 - Returns (random) statistical data

Data Collection Agent Example (optional)

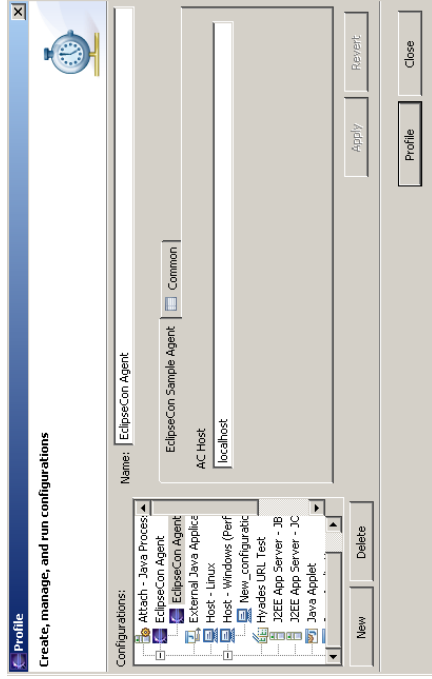
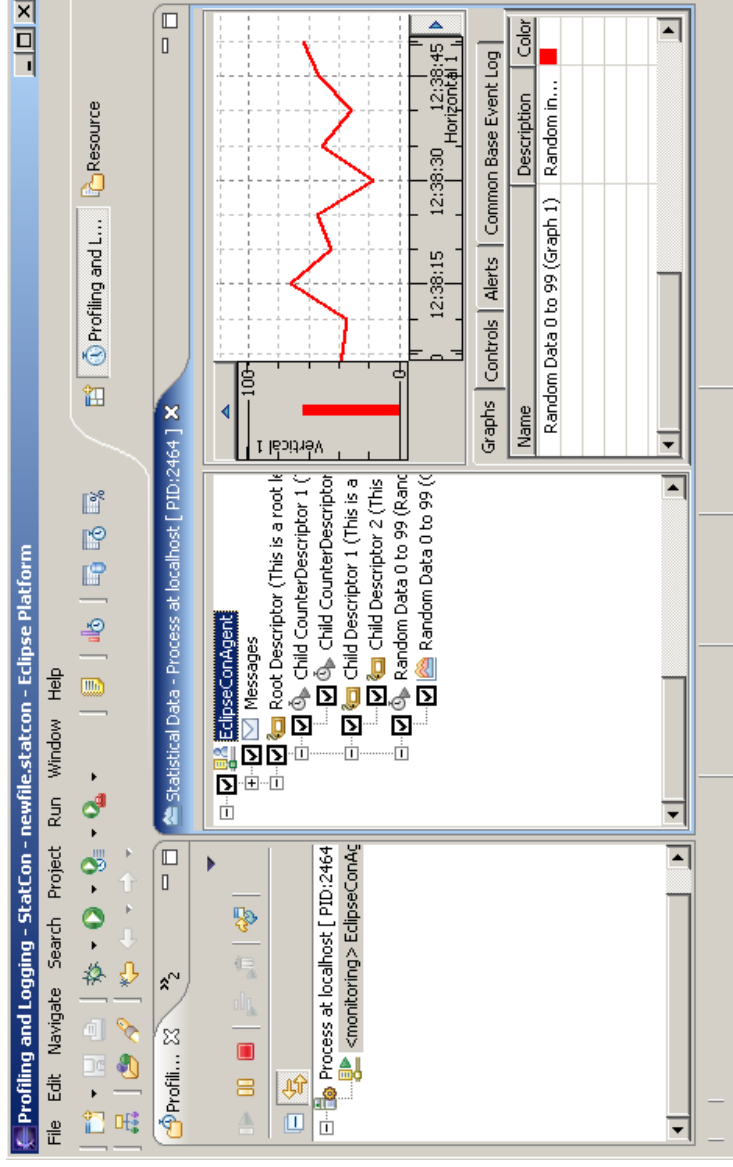
- The AC Agent
 - Extend the class `org.eclipse.hyades.logging.core.LoggingAgent.java`
 - Use `write(String)` to send back fragments
 - Send back fragments of the proper format
 - Package up the agent as an AC plugin and add it to the AC plugins folder (`eclipsecon.tutorial.datacollectionagent` folder in `eclipsecon.tutorial.datacollection.agent` plugin)
- See code in `eclipsecon.tutorial.datacollection.agent` for sample agent

Data Collection Agent Example (optional)

- The Workbench UI
 - Create a new plugin
 - Extend the launch configuration extension points
 - Write a custom launch configuration tab
 - Write a custom launch delegate (agent launcher)
 - Get your launch delegate to create the agent on the AC and set up the models and XML loader

Data Collection Agent Example (optional)

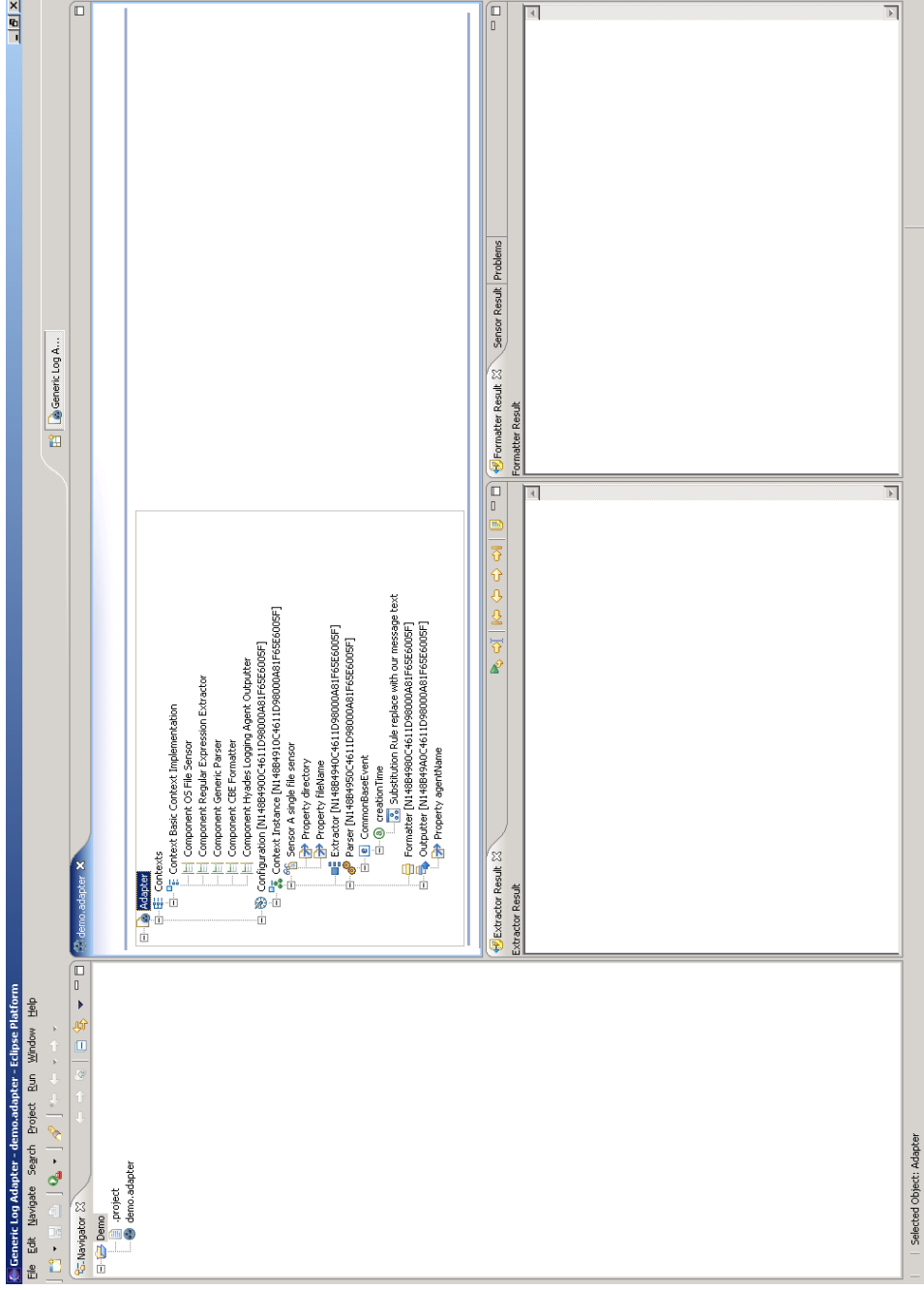
- End result: your very own TPTP data collection agent!



Demonstrations – Logging

- Generic Log Adapter – Creating an Adapter File
 - Create a local Java Logging XML log file based on the content in Appendix 1.
 - Open the Generic Log Adapter Perspective (Window >> Open Perspective >> Other... >> Generic Log Adapter).
 - Create a new simple project (File >> New >> Other... >> Simple >> Project).
 - Create a new adapter configuration file using the Adapter Configuration Editor (File >> New >> Other... >> Generic Log Adapter >> Generic Log Adapter File).
 - Select the local log file as the template log file.

Demonstrations – Logging



Demonstrations – Logging

- Generic Log Adapter – Executing the Adapter File Continuously
 - Add a *SingleFileOutputter* to the Context instance to output resulting Common Base Events to a local log file.
 - Move the following resources to the directory containing the adapter file:
 - hgl.a.jar (org.eclipse.hyades.logging.adapter)
 - hlcbe101.jar (org.eclipse.hyades.logging.core)
 - hlc.core.jar (org.eclipse.hyades.logging.core)
 - hexr.jar (org.eclipse.hyades.execution.remote)
 - ecore.jar (org.eclipse.emf.ecore)
 - common.jar (org.eclipse.emf.common)
 - /schema/* (org.eclipse.hyades.logging.adapter)
 - Run the following command from the directory containing the adapter file:

```
java -DGLA_HOME=<current directory> -classpath
hgl.a.jar;hexr.jar;hlcbe101.jar;ecore.jar;common.jar;hlcore.jar
org.eclipse.hyades.Logging.adapter.Adapter -ac demo.adapter -cc
demo.adapter
```

- Modify the local Java Logging XML log file and view the local Common Base Event log file.

Demonstrations – Logging

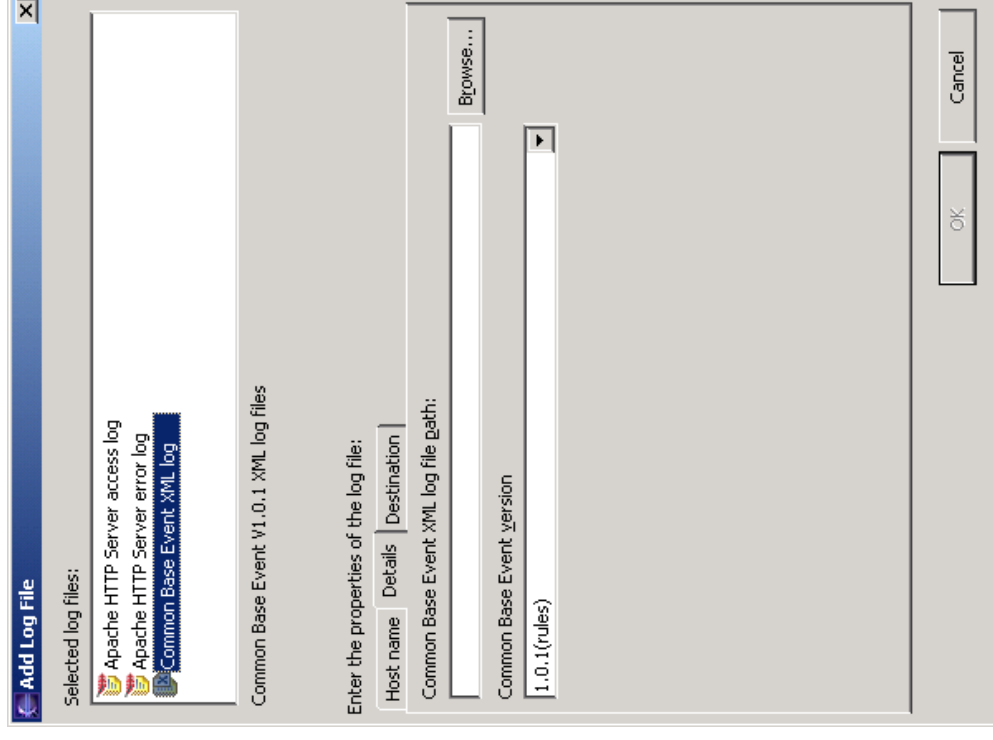
- Logging Scenarios
 - Create a customized Common Base Event Factory (File >> New >> Example... >> Hyades Logging >> Hyades EMF Common Base Event v1.0.1 Sample).
 - Instrument the demo application to natively log Common Base Events (File >> New >> Example... >> Hyades Logging >> Hyades JSR-047 Logging Sample).
 - Configure the demo application's logging facility:
 - Output to a Logging Agent.
 - Common Base Event filtering.
 - XML formatting.
 - [Optional] Write the serialized XML Common Base Events to a Logging Agent (org.eclipse.hyades.logging.core.LoggingAgent) directly.

Demonstrations – Logging

- Log and Trace Analyzer - Monitoring Logging Agents
 - Start the Agent Controller (<Agent Controller install directory>/bin/RAServer.exe).
 - Open the Profiling and Logging Perspective (Window >> Open Perspective >> Other... >> Profiling and Logging).
 - Enable logging in the Profiling and Logging Preferences (Window >> Preferences >> Profiling and Logging).
 - Attach to the demo's Logging Agent (Run >> Profile... >> Attach – Java Process >> New >> Agents).
 - Start monitoring the demo's Logging Agent (<right click the demo's Logging Agent> >> Start Monitoring).
 - [Optional] Attach and start monitoring the Generic Log Adapter's Logging Agent.

Demonstrations – Logging

- Log and Trace Analyzer – Log File Parsers
 - Import the local Common Base Event XML log file (File >> Import... >> Log File >> Add... >> Common Base Event XML log).
 - Create a custom parser to import the Java Logging XML log file using the adapter file (File >> New >> Example... >> Hyades Logging >> Log Parser Sample).
 - Implement the `org.eclipse.hyades.logging.parsers.logParser.extension.point`.
 - Import the local Java Logging XML log file (File >> Import... >> Log File >> Add... >> Simple Parser V1.0).
 - [Optional] Import a remote Java Logging XML log file.
 - [Optional] Import the log Common Base Event log file generated by the Generic Log Adapter.



Demonstrations – Logging

- Log and Trace Analyzer – Log View
 - Select the various imported log files and the Logging Agent in the Log Navigator View.
 - Manipulate the log records in the Log Records pane of the Log View:
 - Navigation.
 - Sorting.
 - Filtering.
 - Searching.
 - Export to Common Base Event XML file.



- [Optional] Create a custom log view to view Common Base Event model data by implementing the `org.eclipse.hyades.ui.analyzerExtensions` extension point.

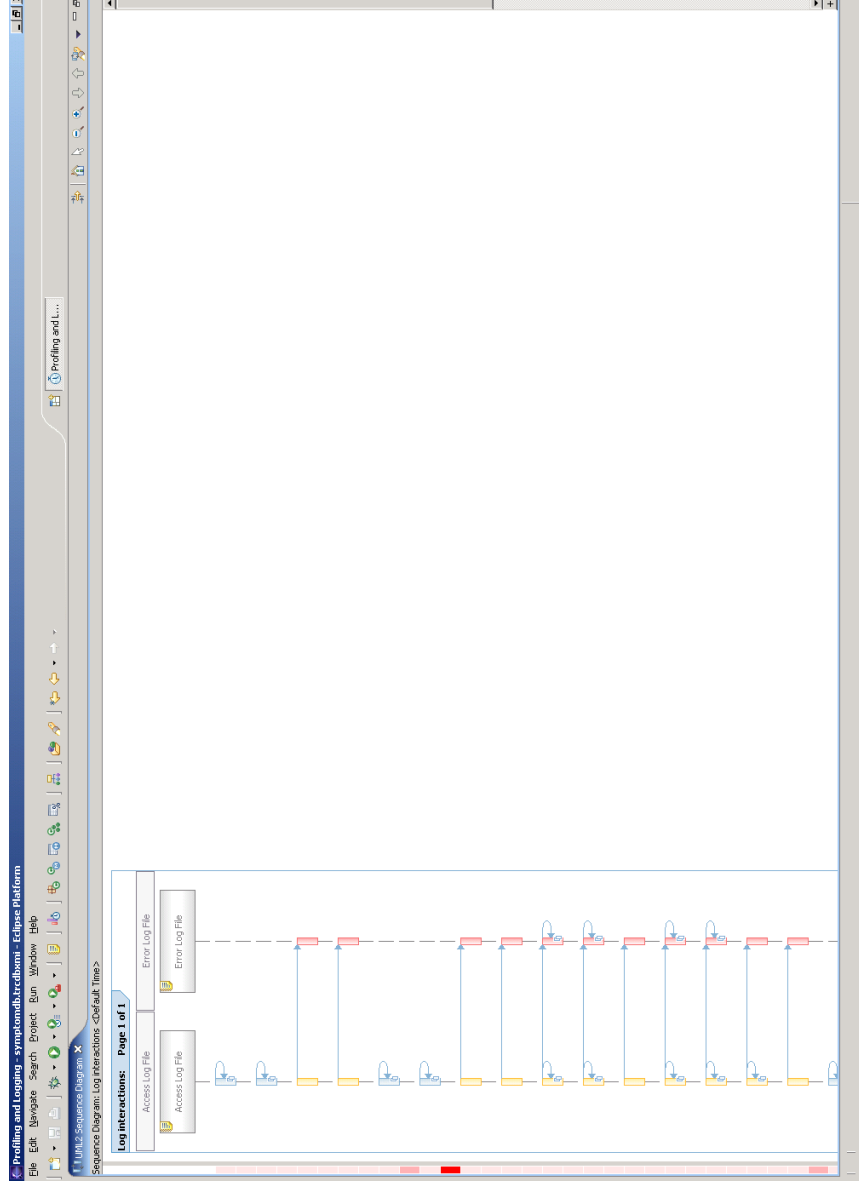
Demonstrations – Logging

- Log and Trace Analyzer - Analysis
 - Create a new symptom database file using the Symptom Database Editor (File >> New >> Other... >> Profiling and Logging >> Symptom Database).
 - Deploy and import the symptom database file (File >> Import... >> Symptom Database File).
 - Analyze the Common Base Event log records in the Log View using the default Log Analyzer (<right click a log record> >> Analyzer >> Default Log Analyzer).
 - Create a customized Analysis Engine (File >> New >> Example... >> Hyades Logging >> Analysis Engine) by implementing the org.eclipse.hyades.analysis.engine.logAnalyzer extension point.
 - Analyze the Common Base Event log records in the Log View using the customized Analysis Engine (<right click a log record> >> Analyzer >> Default Log Analyzer).

Demonstrations – Logging

- Log and Trace Analyzer – Correlation
 - Create a log correlation for the Common Base Event log records in the Log View using the Default Time correlation engine (File >> New >> Other... >> Profiling and Logging >> Log Correlation).
 - View the time-based log correlation in the UML 2 Sequence Diagram View (<right click log correlation> >> Open With >> Log Interactions).
 - Create a customized Correlation Engine (File >> New >> Example... >> Hyades Logging >> Log Correlator Sample) by implementing the org.eclipse.hyades.logc.logInteractionView extension point.
 - Create a log correlation for the Common Base Event log records in the Log View using the customized correlation engine (File >> New >> Other... >> Profiling and Logging >> Log Correlation).
 - View the customized log correlation in the UML 2 Sequence Diagram View (<right click log correlation> >> Open With >> Log Interactions).
 - [Optional] Open the Apache Log Correlation/Analyzer Sample (File >> New >> Example... >> Hyades Logging >> Apache Log Correlation/Analyzer Sample).

Demonstrations – Logging



Conclusion

- TPTP Logging and Monitoring Tools provide a centralized, open-source and extensible mechanism for end-users, administrators, field service engineers and developers to decrease problem determination costs and improve system quality.
- Collects and consolidates log and trace data from disparate systems into a single management tool.
- Common event format for vendor, product and version-specific log and trace formats.
- Consumes common event format for viewing, navigating, sorting, filtering and searching large amounts of log and trace data.
- Correlation to determine a set of related events thereby visualizing control flow across distributed systems.
- Analysis to provide explanation and possible solutions to known problems.

Conclusion - Project Extensibility

Extensible architecture to allow users to define vendor and product specific:

- Agents
 - Views
 - Adapters
 - Parsers
 - Symptom Databases
 - Analysis Engines
 - Correlation Engines
-
- Several existing industry products built on TPTP:
 - IBM Autonomic Computing Tool Kit (<http://www-106.ibm.com/developerworks/autonomic/overview.html>)
 - IBM Rational Functional Tester, Manual Tester, Application Developer and Software Architect (<http://www-306.ibm.com/software/rational/>)
 - Scapa Test and Performance Platform v3.1 (<http://www.scapatech.com/products/intro.html>)

Conclusion - Future Work

- 3.3 Planned Work
 - C implementation of the Common Base Event v1.0.1 specification.
- 4.0 Planned Work
 - Agent Controller will provide well defined APIs for agents to export control variables to allow users to monitor and change the environment
 - Statistical Data view and StatCon editor will be broken into a set of views allowing easy extension for future views (e.g. pie charts).
- Future Work
 - CBE 2.x implementation.
 - Eclipse Common Logging

Conclusion - Resources

- Bridgewater, D., “Standardize messages with the Common Base Event model”, <ftp://www6.software.ibm.com/software/developer/library/ac-cbe1.pdf>.
- Eclipse Modeling Framework (EMF), <http://www.eclipse.org/emf>.
- TPTP Documentation (Help >> Help Contents).
- TPTP Examples (File >> New >> Example...).

Conclusion - Resources

- TPTP Project, <http://www.eclipse.org/TPTP> (formally <http://www.eclipse.org/hyades>).
 - Project architecture, organization and plans.
 - Downloads.
 - Defects.
 - CVS.
 - Developer Resources.
 - Documentation:
 - Common Base Event v1.0.1 specification, schema and API documentation.
 - Tutorials.
 - Mailing Lists.
 - Newsgroup.

Questions, Answers and Discussion

- Is there a *real* requirement in industry for logging and monitoring tooling?
- What are some common PD activities?
- Are there viable proprietary extensions to the TPTP logging and monitoring tooling?
- Possible suggestions for enhancements for the existing TPTP logging and monitoring tooling.

Appendix 1 – Java Logging XML Log File

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE log SYSTEM "logger.dtd">
<log>
  <record>
    <date>2005-02-28 13:30:01</date>
    <millis>1109615401123</millis>
    <sequence>1234</sequence>
    <logger>org.eclipse.hyades.DemoLogger</logger>
    <level>INFO</level>
    <class>org.eclipse.hyades.DemoClass</class>
    <method>main</method>
    <thread>1</thread>
    <message>Welcome to Tutorial #22</message>
  </record>
</log>
```

Appendix 2 – logger.dtd

```
<!ELEMENT log (record*)>
<!ELEMENT record (date, millis, sequence, logger?, level, class?,
    method?, thread?, message, key?, catalog?, param*, exception?)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT millis (#PCDATA)>
<!ELEMENT sequence (#PCDATA)>
<!ELEMENT logger (#PCDATA)>
<!ELEMENT level (#PCDATA)>
<!ELEMENT class (#PCDATA)>
<!ELEMENT method (#PCDATA)>
<!ELEMENT thread (#PCDATA)>
<!ELEMENT message (#PCDATA)>
<!ELEMENT key (#PCDATA)>
<!ELEMENT catalog (#PCDATA)>
<!ELEMENT param (#PCDATA)>
<!ELEMENT exception (message?, frame+)>
<!ELEMENT frame (class, method, line?)>
<!ELEMENT line (#PCDATA)>
```