Profiling Java Applications Running on Embedded Devices with Restricted Resources

Dr. Dimitar Valtchev, CTO, ProSyst Software GmbH

Eclipse Embedded Day
Stuttgart Jun 25, 2009
www.microdoc.com/eday

Agenda

- Introduction and problem description
- Architecture and main features of ProSyst (m)JProfiler
- Short demo
- Use case – profiling of mobile devices with OSGi stack
- Summary

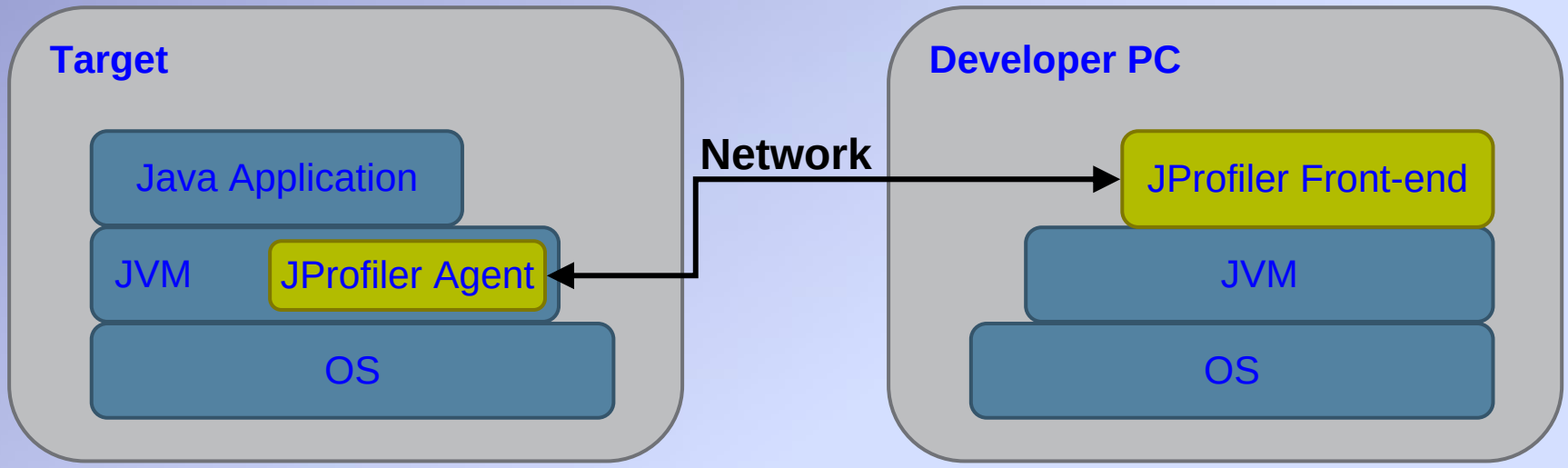
Introduction

- Nowadays there are several popular Java profilers which work very well with J2SE and J2EE JVMs
- They support:
  - Local and remote profiling
  - Memory profiling
  - CPU profiling
  - Visual representation and statistics (VM load, GC activities, etc.)
- Profiling Java applications on resource-constrained mobile and embedded devices is not possible/efficient with the existing tools

Problem Description

- Profiling on a emulator is rarely helpful because there is a big difference in application behavior when running on an emulator and when running on the mobile device regarding:
  - Memory usage
  - CPU usage
  - Storage devices (e.g. flash)
- Profiling on the target system is often not possible because of the missing support for numerous combinations OS/JVM
- Even if the JVM on the target system can be run in profiling mode , the behavior of the applications is often influenced significantly by the restricted resources
- The standard profilers does not taken into account some well established architectures used for Java embedded systems - OSGi

JProfiler Architecture



Legend: **ProSyst components**

- Solution: Optimized profiling agent which communicates actively with the profiling environment

Profiling Agent Requirements

- Minimal impact on the application – shall consume minimal system resources and shall not influence the application behaviour
- Low memory consumption of the profiler agent
- Portability
- Convenient deployment of the profiling agent and easy configuration

Implementation Details 1(2)

- Buffers the information per thread
  - Does not need locking except when sending the buffer
  - Can send multi-records (reduces the traffic because the type and thread is the same and is sent only once)
- Uses RLE encoding
  - 40% reduction in traffic
- Memory is allocated in chunks
  - Limits the overhead from small allocations
  - Prevents fragmentation
- Customized hash tables for different record types (depending on how well they hash)

Implementation Details 2(2)

- Only bare minimum of information is kept in the agent, all is sent for processing to the front-end
  - Storing the information inside the agent is in most cases slower than sending, not to mention the increased memory foot-print
- All machine dependent code is placed in separate modules
  - It is easier to port to another platform
  - Avoids the hard maintenance of “ifdefs”
  - Includes only raw methods for working with sockets, threads and time
  - If possible, methods from JVMPI are used

ProSyst (m)JProfiler Key Features 1(3)

- (m)JProfiler is implemented as a plug-in for Eclipse
- Local and remote profiling
- Visual representation of the VM load in terms of:
  - Active and total bytes
  - Instances
  - Threads
  - Classes
  - Garbage Collector activity

ProSyst (m)JProfiler Key Features 2(3)

- Memory and CPU profiling
- Two methods for CPU profiling:
  - Timing method: Measures the time it takes to invoke a Java method and the number of method calls.
  - Sampling method: Tracks the threads activity over a specified interval of time.

ProSyst (m)JProfiler Key Features 3(3)

- Full thread dump and deadlock detection
- Threads activity information
- Trigger support
- Tracing heap information and memory stack frames
- Incoming and outgoing references
- Garbage collection during profiling

Supported Platforms (standard installation)

OS	Processor
Windows CE	ARM
Linux	x86
Linux	MIPS (Big and Little Endian)

Other Supported Platforms

OS	Processor
Net BSD	x86
QNX	SH4
VxWorks	Power PC

Supported Java Virtual Machines

- Sun JDK
- IBM JDK
- Oracle JRockit
- IBM J9
- Sun CVM
- Skelmir CEE-J
- Aonix PERC
- Esmertec Jeode

Short Demo

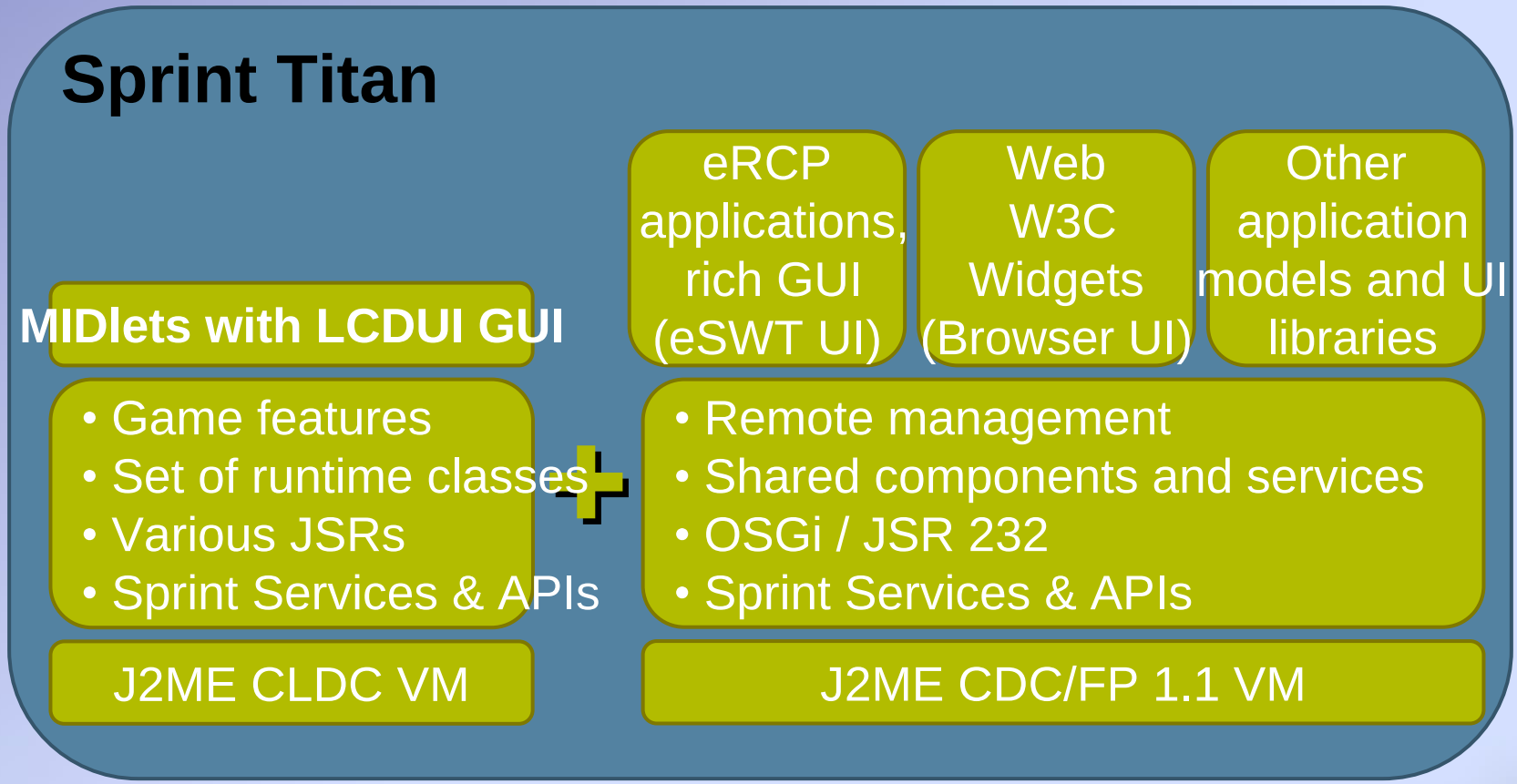
Use case – profiling of mobile devices running OSGi stack (Sprint/Titan or Nokia eRCP)

- Why OSGi for Mobile/Embedded Devices? Why not simply use CDC as it is?
- Because it does not provide …
  - a dynamic component model runtime & APIs
  - support for existing application models (like MIDP, eRCP, etc.)
  - an application focused security model
  - base services as configuration, logging, etc.
  - device management and deployment APIs

OSGi Architecture for Mobile Application

- OSGi Mobile specification became part of the JCP as JSR 232: Mobile Operational Management.
- JSR 232 Mobile Extensions to OSGi:
  - Generic support for multiple application models (eg. MIDP, Xlets, DoJa, etc.)
  - A standardized deployment and packaging model
  - Auto-configuration of bundles and packages
  - A standardized device management model for the Java environment, aligned to OMA-DM Standard
  - Monitoring API
  - Standard condition classes for mobile phones

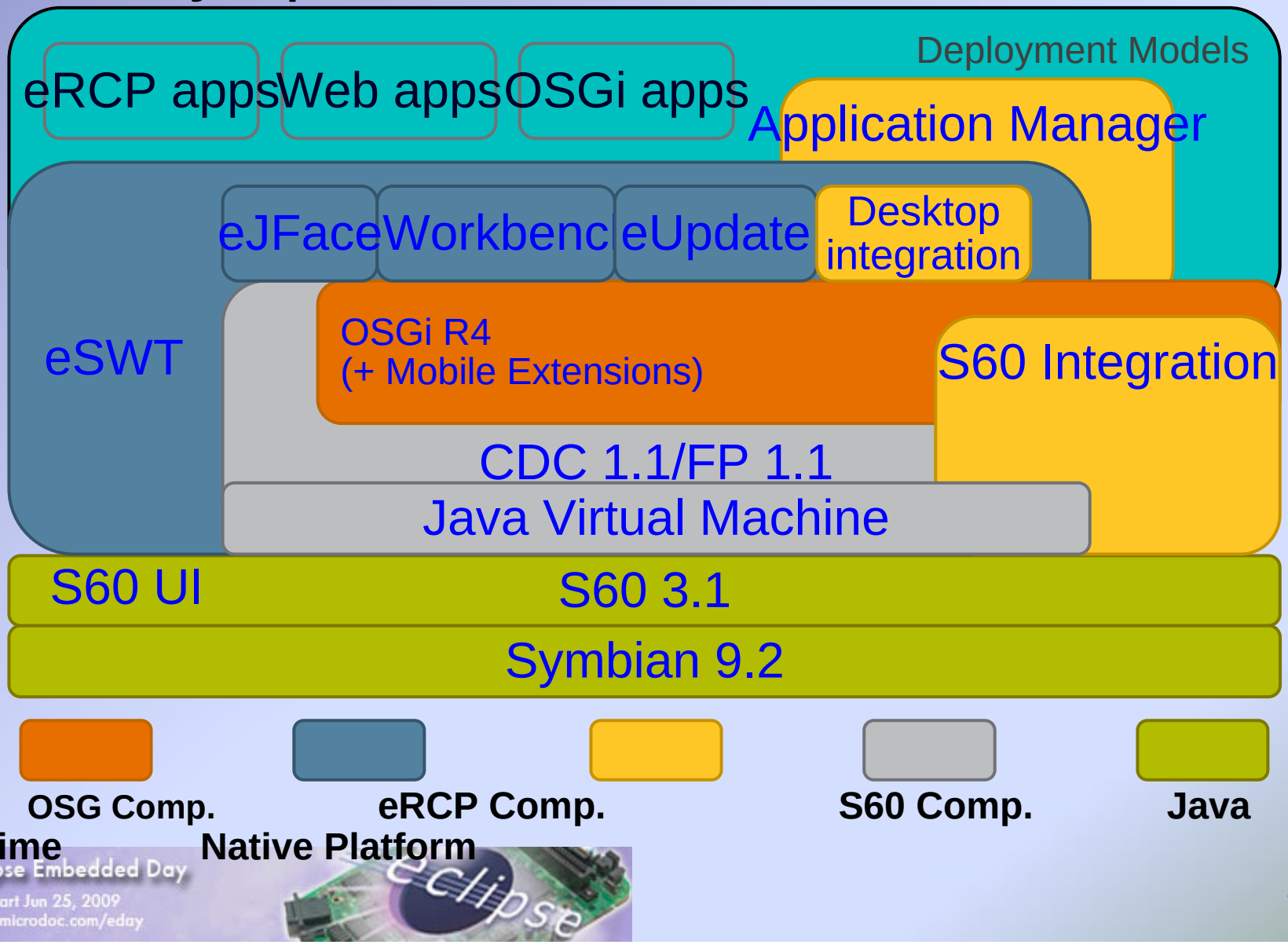
Industry uptake: Sprint Titan Mobile Java



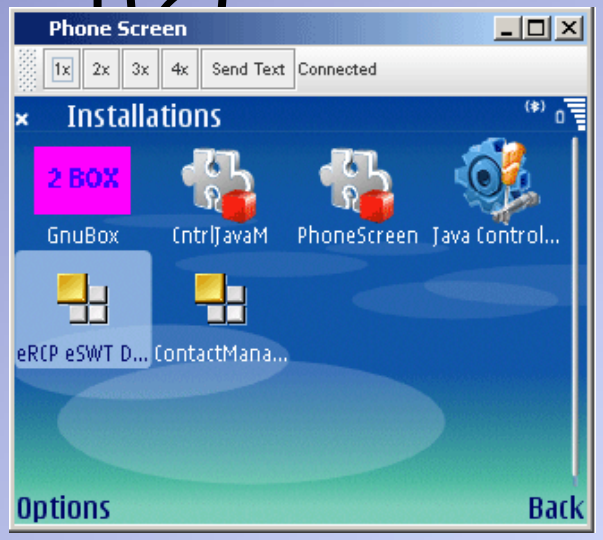
"The most suitable cross-platform runtime for use in mobile handsets"

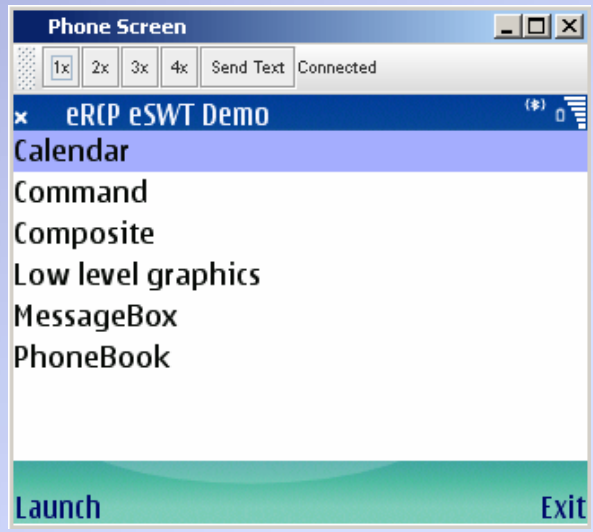
- Source: Sprint Developer Site: http://developer.sprint.com

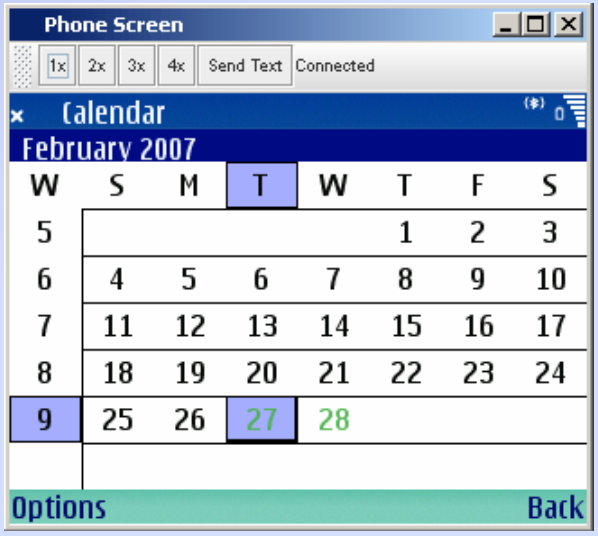
Industry uptake: Nokia eRCP for S60



Symbian UI Integration 1(2)



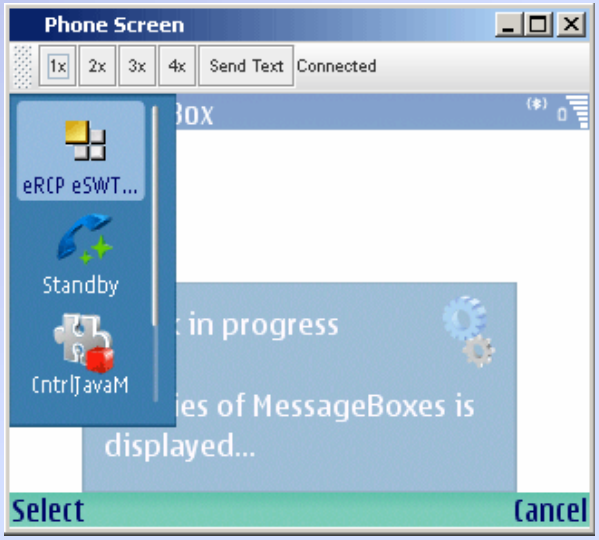


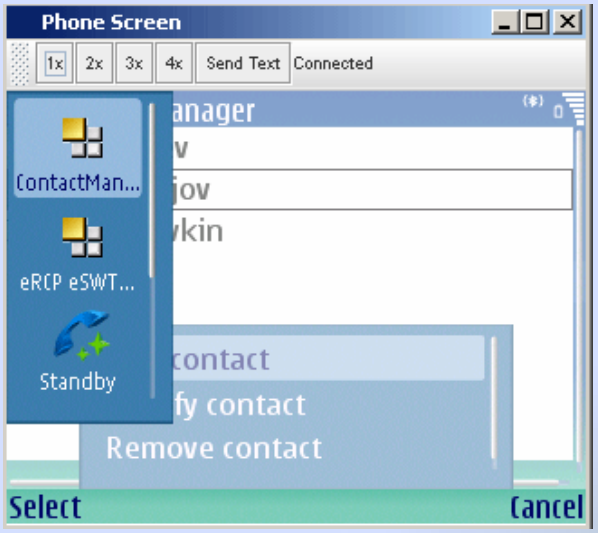


- Full integration with the Symbian look & feel. eSWT provides Symbian-like components.
- Icons for all installed eSWT applications are shown on the desktop.
- There is no difference in the behaviour of Symbian and Java applications. In most cases, the start of the Java applications is even faster.

Symbian UI Integration 2(2)

- Switching between any applications including eRCP by using the task list of the phone.
- The example below shows the started eRCP eSWT Demo application and the Contact Manager application. Both applications appear in the task list of the phone and the task list can be used for switching between them.





CPU and Memory Statistics (Nokia S60)

- [Mem stats]
  - Used memory: (instant) 378980 bytes , (peak) 378980 bytes
  - Used blocks: (instant) 77, (peak) 77
- [IO stats]
  - Send data: 90607751 bytes
  - Send time: 9380 ms
  - Average speed: 9659674 b/s
- [CPU stats]
  - METHOD_ENTRY - Number of calls: 11379785
  - METHOD_EXIT - Number of calls: 11379767
  - OBJECT_ALLOC - Number of calls: 724705
  - OBJECT_FREE - Number of calls: 672304
  - OBJECT_MOVE - Number of calls: 315183
  - …

CPU and Memory Overview

- 370 KB Used Memory
- Send Data:
  - Memory & CPU profiling: 90 MB (9.6 MB/s)
  - Memory profiling: 74 MB (6.9 MB/s)
  - CPU profiling: 14 MB (296 KB/s)
- CPU Statistics
  - Memory & CPU profiling: 11 Mio. method calls, 724 k object alloc
  - Memory profiling: 524 k object alloc
  - CPU profiling: 5 Mio. method calls

Summary

- Profiling of embedded Java applications has special requirements to the profiling agent and the IDE.
- The used memory and the influence on the applications being profiled can be significantly reduced by increasing and optimizing the data stream sent to the profiling application.
- Such an approach is applied in the existing ProSyst's (m)JProfiler.
- ProSyst (m)JProfiler is used successfully in profiling many embedded applications and is for example standard part of Titan Developer Tools.

Thank you for your attention!

- Dr. Dimitar Valtchev, CTO, ProSyst Software GmbH
  d.valtchev@prosyst.com

- ProSyst Products
  http://www.prosyst.com/products/tools.html
- ProSyst Developer's Zone
  http://dz.prosyst.com
- Sprint Titan Platform
  https://developer.sprint.com/site/global/develop/technologies/sprint_titan/p_sprint_titan.jsp