



Usage monitoring of your Java applications during runtime

Paul René Jørgensen (Telenor) & Steinar Cook (Balder Programvare)

The problem

What's
going on?



The problem cont.

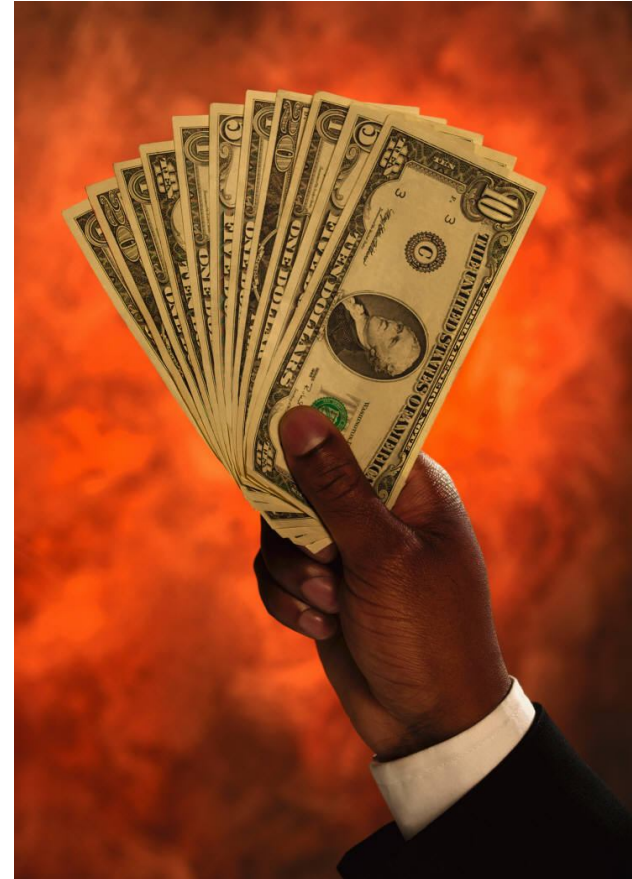
- Traffic volume per:
 - Application server, cluster, package, class, method, principal per time unit?
- Runtime dependencies?
- Applications or classes no longer in use?
- Who uses how much CPU?
- Irregular usage of exceptions?
- Server load and balancing?

Even more questions!

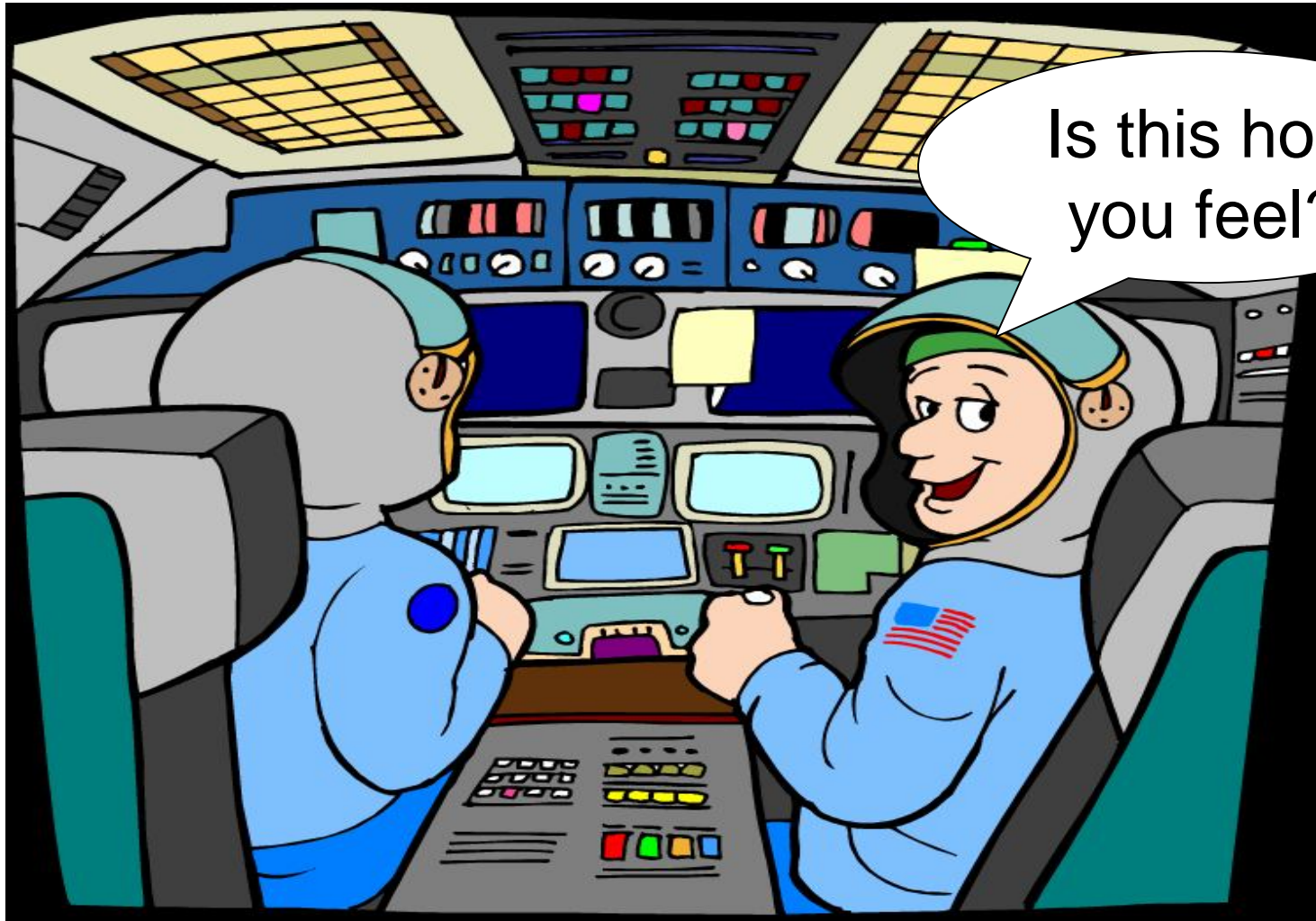
- Differences between versions of my application?
- Installing applications; into which cluster?
- What are the main paths through the entire system?
- If I change this application, who will be affected?
- Which class does my class invoke?
- What is the response time for a given service?

Current solutions?

- Many solutions in the market
 - Jconsole
 - HP (Mercury)
 - Performasure / Foglight
 - YourKit
 - jProfiler
- But, they are;
 - Costly
 - Proprietary
 - Complicated!

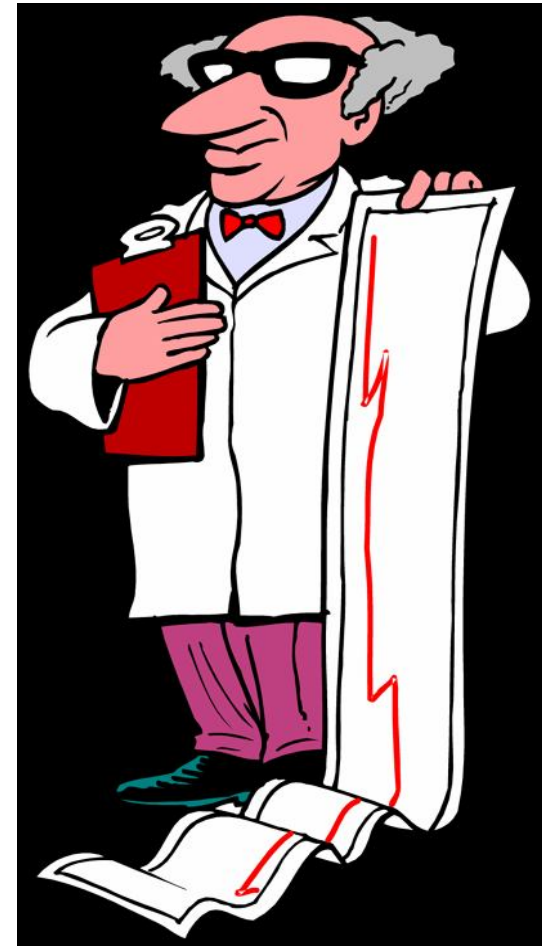


Difficult to use!



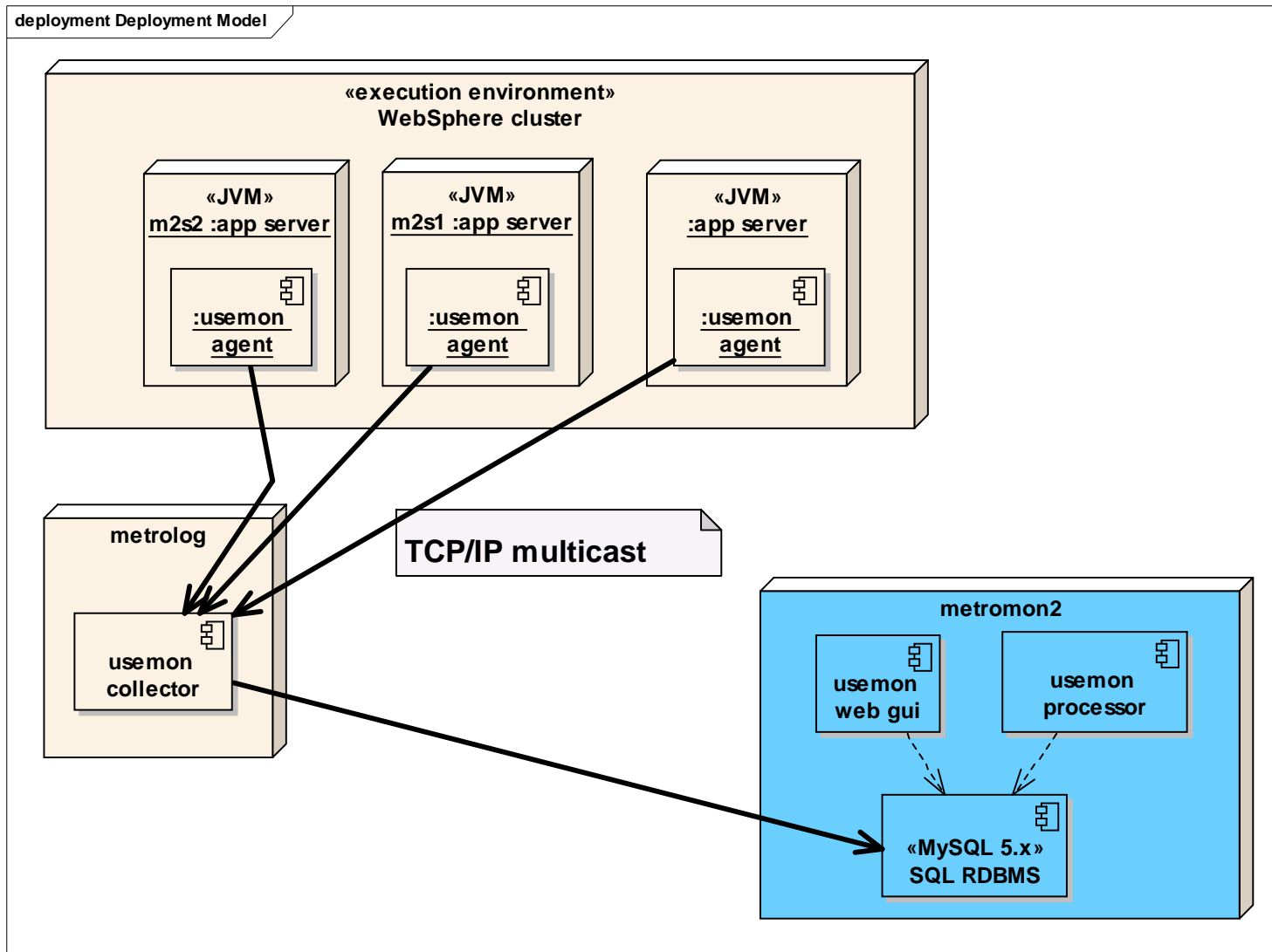
Our solution - KISS

- Simple as possible, but no simpler!
- Harvest data with an agent
- Collect raw data into SQL database
- Data model with OLAP/BI capabilities
- GUI of choice



UseMon beta

Technical overview



The Usemon Agent \leq JDK 1.4

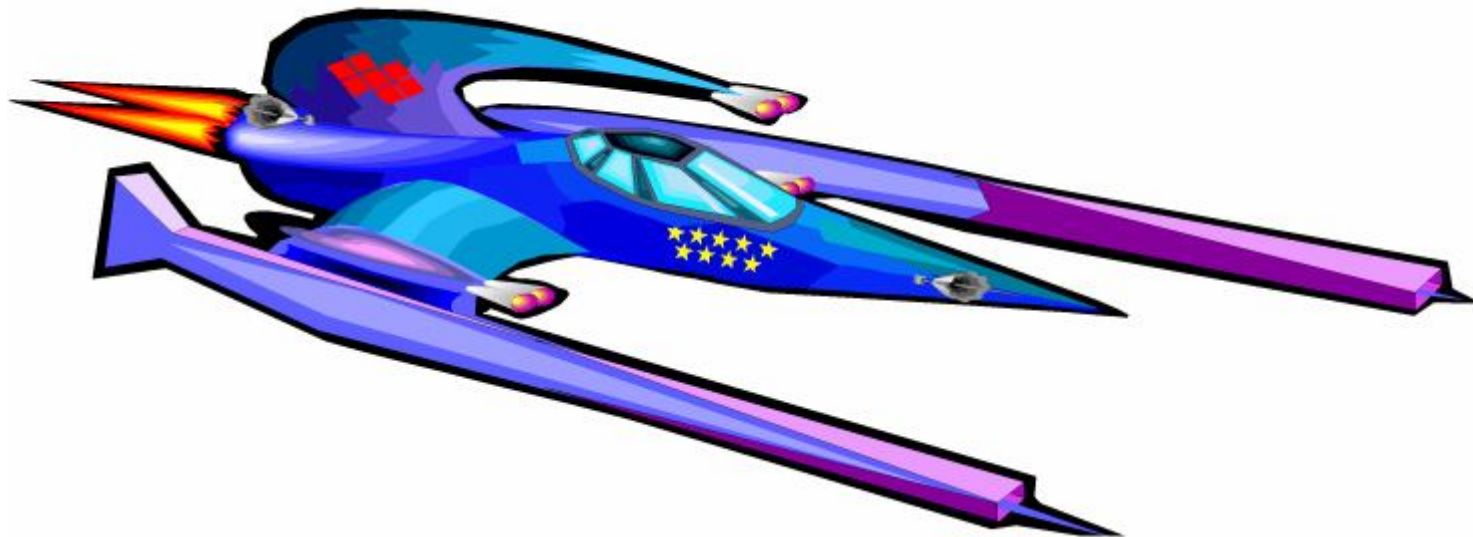
1. Modify `java.lang.ClassLoader`
2. Modified class loader prepended to bootstrap class path
3. Public methods modified
 - EJBs, Servlets, MDBs, POJOs (custom declared)
4. Observations are multicasted



UseMon beta

The Usemon agent \geq JDK 1.5

- Java Agent API for bootstrap
- Interception of public methods through byte code manipulation like JDK \leq 1.4



UseMon_{beta}

The non-intrusive agent

- Yields if application threads
 - Uses excessive memory
 - Heap is growing too large



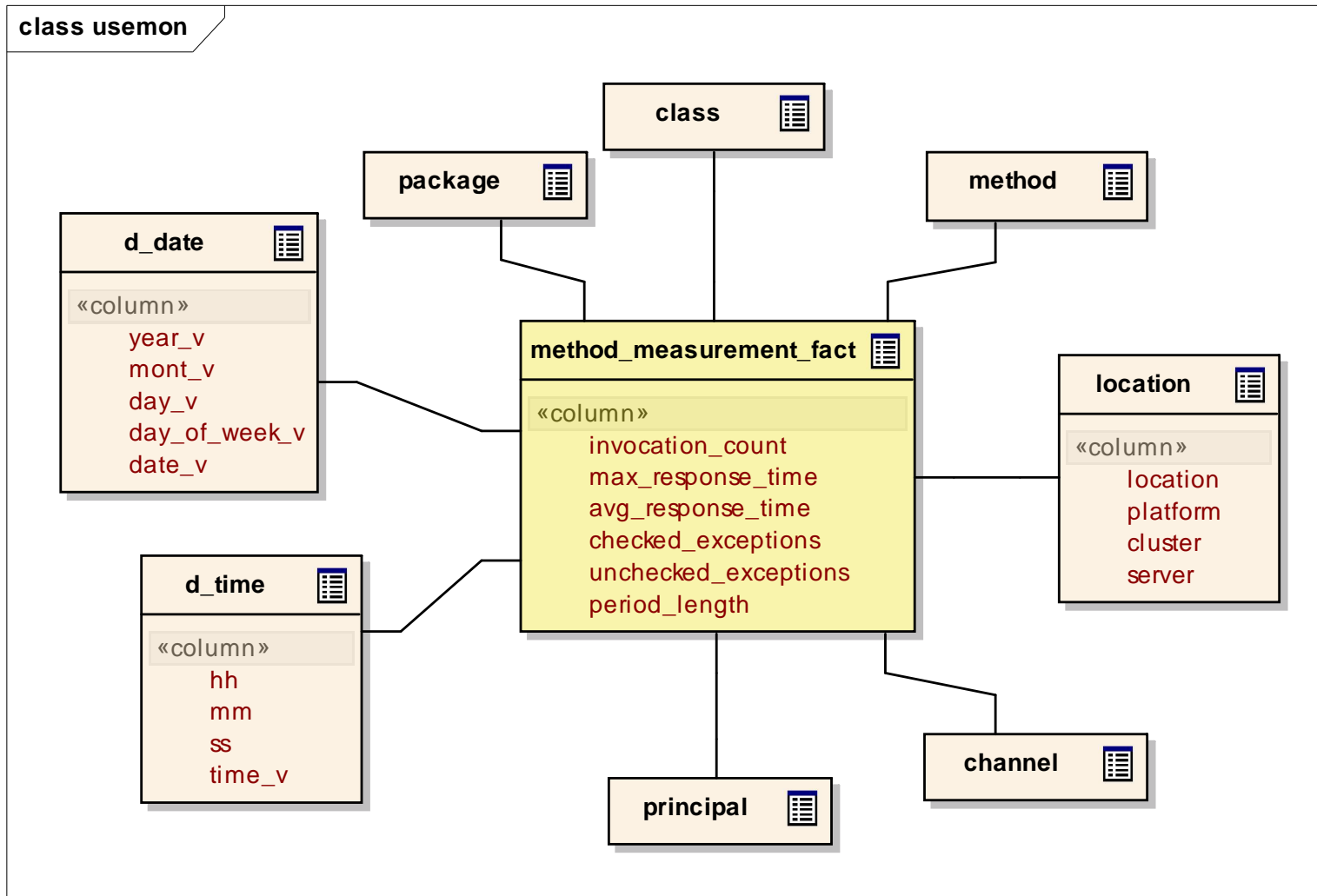
The Usemon Collector

- Receives multicasted observations from agent
- Stores data into SQL database
- Drops observations if free memory falls below threshold of 5% in the Collector JVM (Java Virtual Machine)
- JMX enabled for monitoring and management

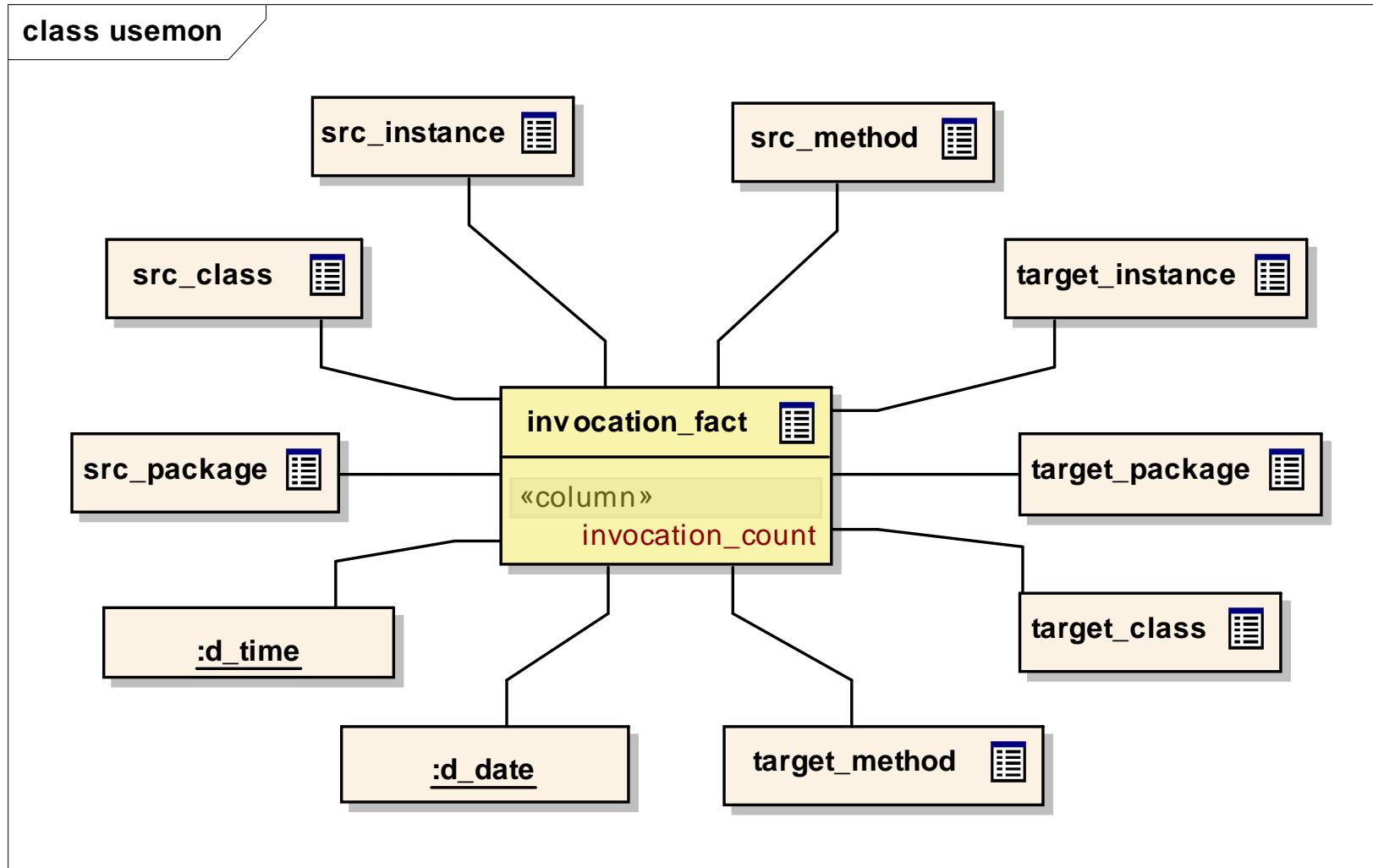
Data Model

- Method observation (statistics) per minute
 - Invocation counter
 - Maximum response time
 - Average response time
 - Checked / Unchecked Exceptions
- Invocation observation (call graph)
 - Source
 - Target
- Heap observations
 - Free, total and maximum memory
 - Derivates : Used memory

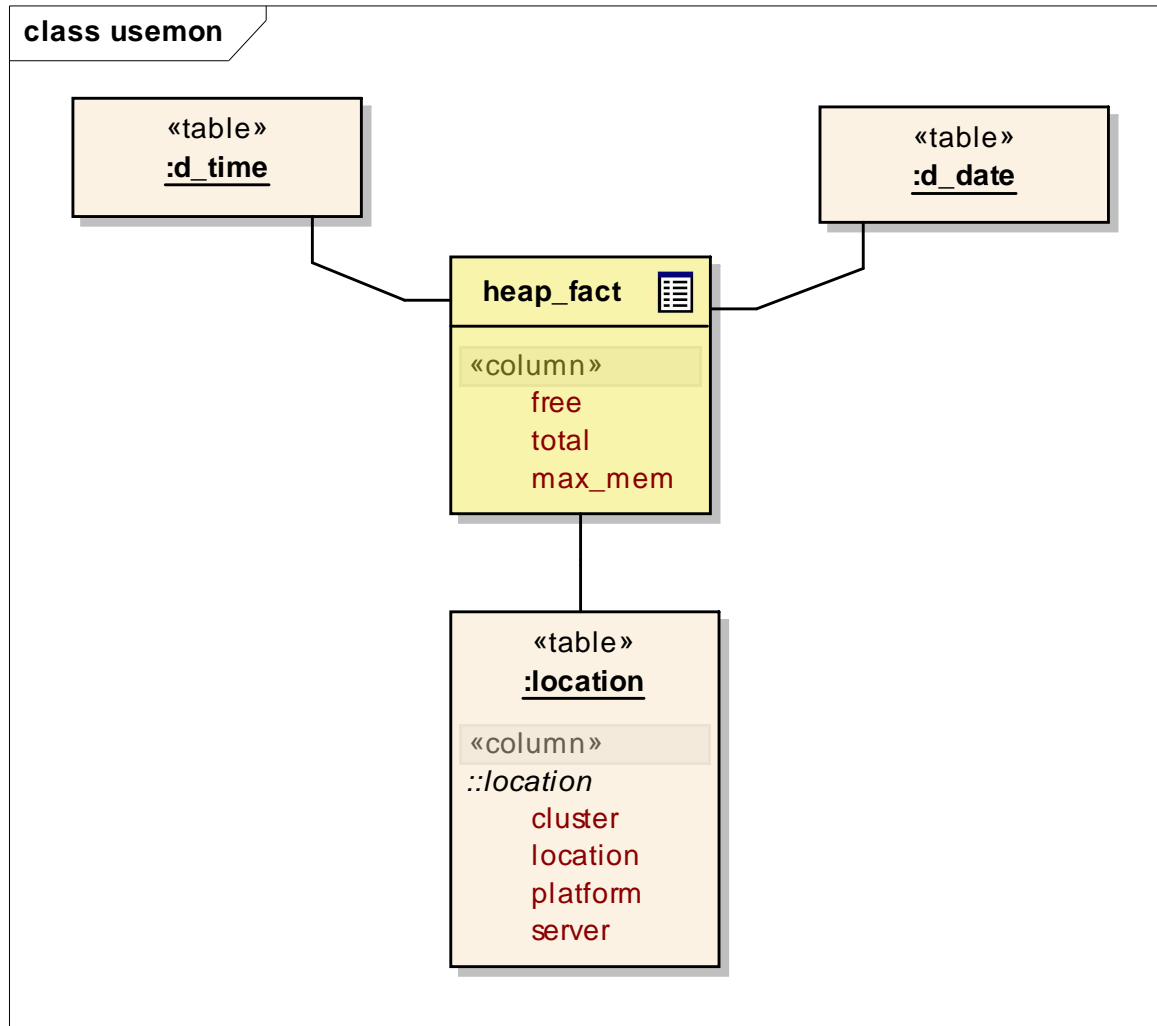
Method statistics observation



Invocation observation

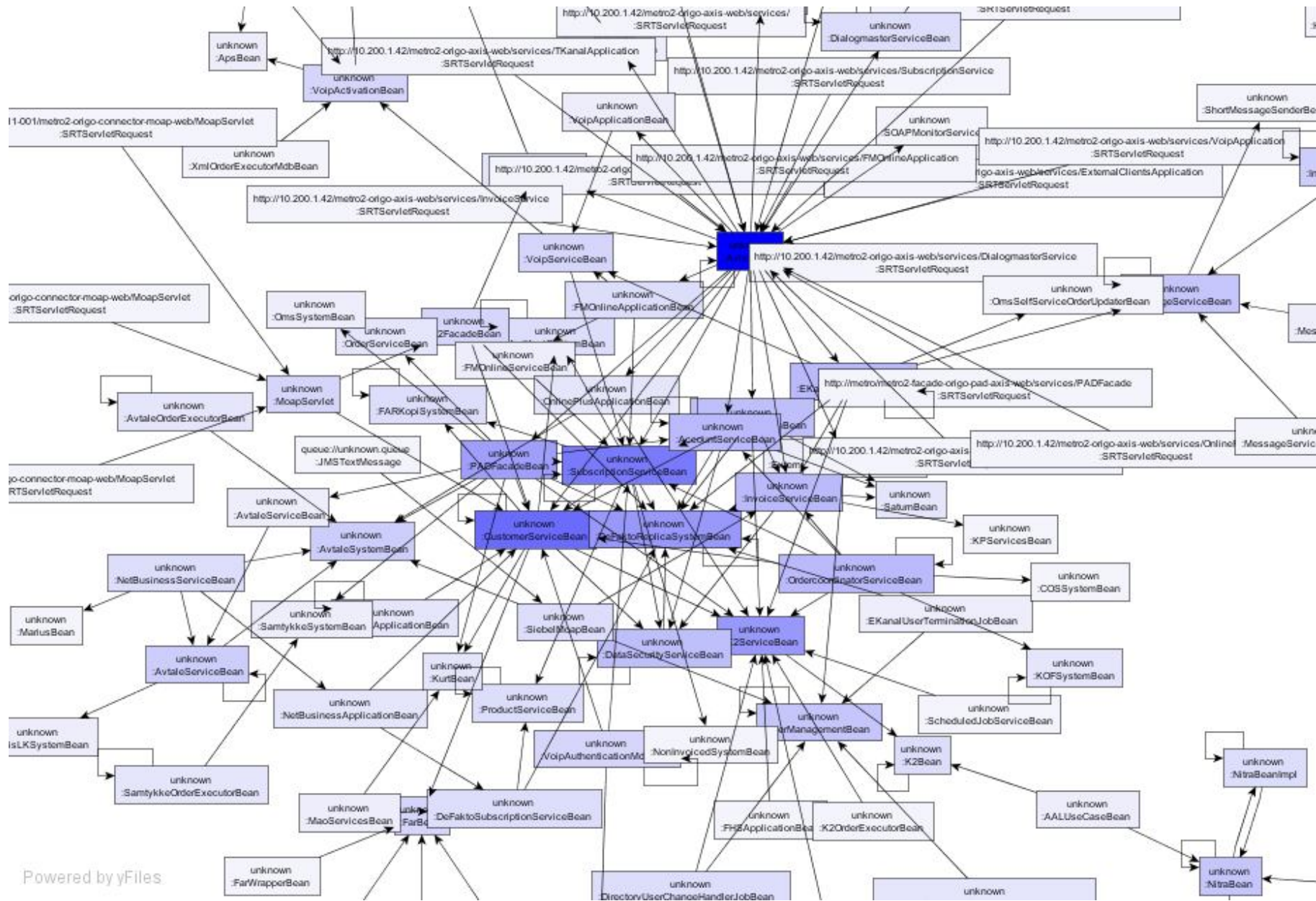


Heap observation



SQL examples – “live demo”

- Latest 30 method observations
- Which method is invoked by a given method



Powered by yFiles

Demo : Dependency graph

UseMon beta

OLAP / BI

- Several OLAP/BI tools available
 - Mostly commercial and expensive
 - A few OSS tools based upon Mondrian
- Requires detailed knowledge of the OLAP cube model and MDX query language
 - Cube model based upon Usemon dimensional model
- Demo : Jasper Server & Jasper Analysis
 - Data from February and specific samples from February 12th (2008)

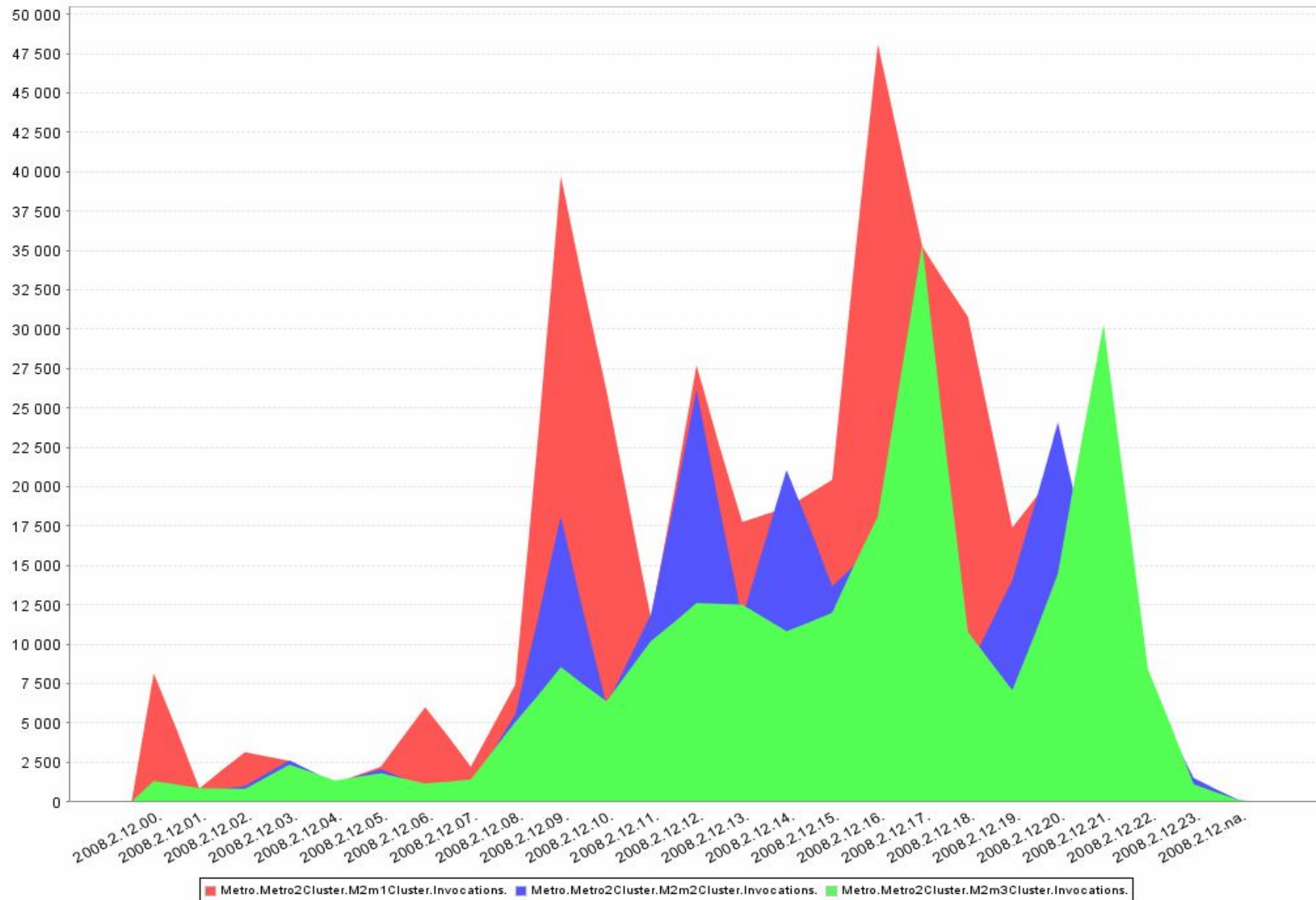
MDX Example Query

```
select
  crossjoin(
    crossjoin(
      {[Location].[Metro].[Metro2Cluster]},
      [Location].[Metro].[Metro2Cluster].children
    ),
    {[Invocations], [Average Response Time]}
  ) on columns,
non empty
  crossjoin(
    {[Date].[2008].[2].[12]},
    [Time].children
  ) on rows
from Usemon
```

Invocations & Response Time per Server over time

		Location					
		+Metro2Cluster					
		M2m1Cluster		M2m2Cluster		M2m3Cluster	
		Measures		Measures		Measures	
Date	Time	• Invocations	• Average Response Time	• Invocations	• Average Response Time	• Invocations	• Average Response Time
12	+00	8148	635.38 ms	1089	21.42 ms	1311	14.52 ms
	+01	839	100.58 ms	586	66.43 ms	852	72.87 ms
	+02	3141	3737.17 ms	1003	19.53 ms	815	43.33 ms
	+03	2589	64.21 ms	2623	84.83 ms	2350	51.66 ms
	+04	1143	43.43 ms	1153	44.33 ms	1335	35.90 ms
	+05	2219	71.27 ms	2071	78.51 ms	1804	100.53 ms
	+06	6003	87.99 ms	934	10.58 ms	1155	10.33 ms
	+07	2214	40.65 ms	909	18.62 ms	1401	34.18 ms
	+08	7414	946.23 ms	5531	293.86 ms	5049	105.63 ms
	+09	39710	1539.25 ms	18117	327.07 ms	8544	129.75 ms
	+10	26112	1292.45 ms	6162	40.72 ms	6373	153.65 ms
	+11	11681	415.62 ms	11883	43.07 ms	10190	102.24 ms
	+12	27714	761.73 ms	26161	246.67 ms	12612	439.56 ms
	+13	17758	169.95 ms	11668	95.45 ms	12511	96.23 ms
	+14	18679	162.08 ms	21063	72.70 ms	10821	146.54 ms
	+15	20451	1039.13 ms	13669	55.96 ms	11994	89.53 ms
	+16	48097	1411.50 ms	16363	214.88 ms	18124	72.63 ms
	+17	35261	1183.84 ms	18167	89.66 ms	35386	173.81 ms
	+18	30796	46.92 ms	8373	35.75 ms	10768	61.67 ms
	+19	17409	207.06 ms	14058	72.78 ms	7071	86.82 ms
	+20	21210	51.39 ms	24093	30.58 ms	14493	45.51 ms
	+21	11493	19.47 ms	11727	23.15 ms	30305	12.20 ms
	+22	7497	12.93 ms	7076	16.76 ms	8391	18.43 ms
	+23	1305	1051.34 ms	1506	17.56 ms	1103	17.60 ms
+na		69	7.75 ms	83	10.92 ms	83	6.33 ms

Invocations per server over time



K2Bean Usage (Random sample)

Date			Measures						
Year	Month	Day	• Invocations	• Average Response Time	• Max Response Time	• Checked Exceptions	• Unchecked Exceptions	• Exceptional Exits	• CPU Time
2008	2	1	1210	1187.64 ms	40786.00 ms	0	0	0.00%	0.40h
2008	2	10	4516	838.51 ms	34151.00 ms	0	0	0.00%	1.05h
2008	2	11	41499	774.19 ms	64943.00 ms	56	0	0.13%	8.92h
2008	2	12	35490	1474.48 ms	59899.00 ms	6	0	0.02%	14.54h
2008	2	13	24683	1004.06 ms	45262.00 ms	0	0	0.00%	6.88h
2008	2	5	13623	1169.85 ms	44330.00 ms	7	0	0.05%	4.43h
2008	2	6	25544	1155.61 ms	67350.00 ms	109	1	0.43%	8.20h
2008	2	7	8028	1127.57 ms	54638.00 ms	16	0	0.20%	2.51h
2008	2	8	8764	742.05 ms	28090.00 ms	20	0	0.23%	1.81h
2008	2	9	633	136.43 ms	18339.00 ms	0	0	0.00%	0.02h

Slicer: [Class=K2Bean]

Top Thread time users (>0.1h)

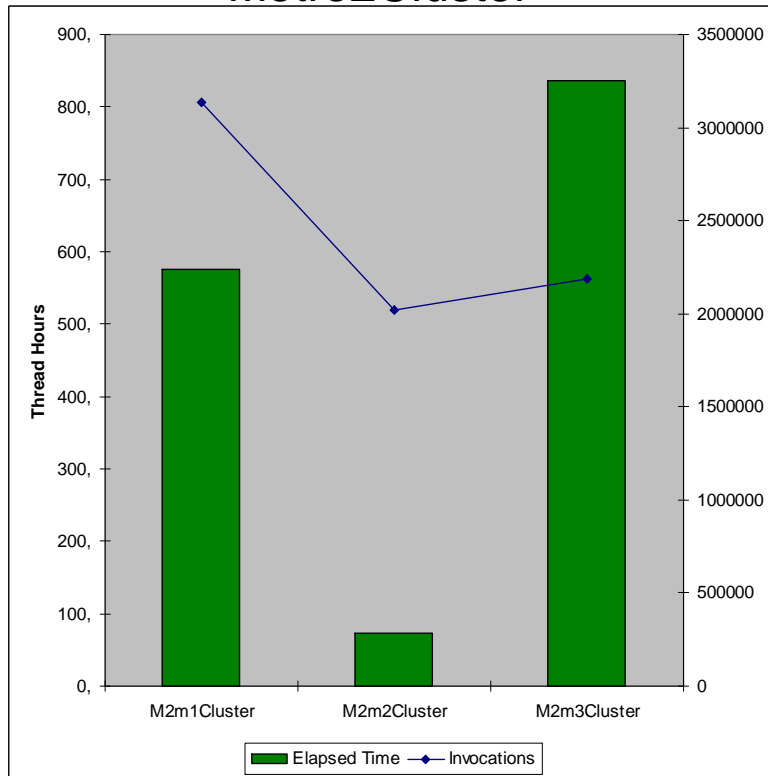
		Measures		
Class	Method	• Invocations	• CPU Time	• Exceptional Exits
AxisServlet	doPost	2675	0.35h	0.00%
AxisServletBase	service	2675	0.35h	0.00%
CustomerServiceBean	getCustomers	10450	0.17h	0.00%
DeFaktoReplicaSystemBean	sp_getSubscrsForCustP	15	0.16h	0.00%
	sp_getTerminatedDirNums	1	0.24h	0.00%
DirectoryUserChangeHandlerJobBean	createInput	157	0.39h	0.00%
	insertChanges	154	0.15h	0.00%
FieldEngineerResponseServlet	doGet	987	0.22h	0.00%
	doPost	987	0.22h	0.00%
FlowSystemBean	receiveEconomicFeedback	944	0.16h	0.00%
InstallerSystemBean	sendEconomicFeedbackToFlow	1888	0.32h	0.00%
InstalledBaseCustomerHandlerJobBean	insertChanges	84	2.18h	0.00%
InstalledBaseSystemBean	sp_insertCPR	1057	0.13h	0.00%
	sp_updateIBCustomerInformation	489	4.49h	0.00%
InstalledBaseUserHandlerJobBean	createInput	105	0.23h	0.00%
	insertChanges	104	0.16h	0.00%
SubscriptionServiceBean	getSubscriptionsForCustomer	12	0.16h	0.00%
	mapCmKofSubscriptionToSubscription	10436	0.17h	0.00%

Top Exception throwers (>0.5%)

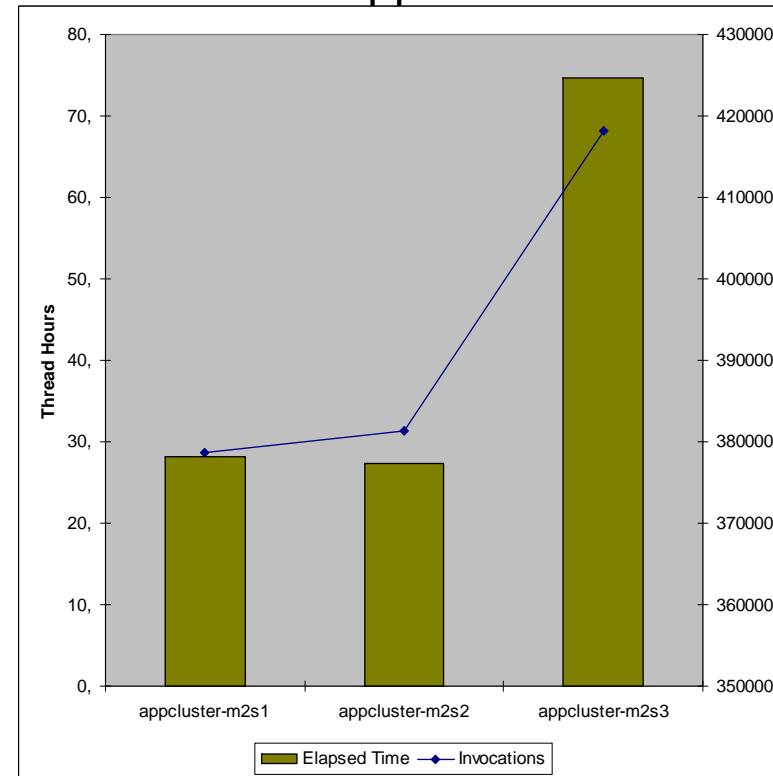
Class	Method	Measures		
		• Invocations	• CPU Time	▼ Exceptional Exits
OrdercoordinatorServiceBean	k2Search	1	0.00h	100.00%
SaturnBean	checkApproveInstalment	168	0.00h	99.40%
InvoiceServiceBean	checkInstallmentApprovalAcceptReminder	158	0.00h	99.37%
TkanalApplicationBean	getNonInvoiceBalance	7	0.00h	85.71%
	getAccountStatus	11	0.00h	63.64%
	getInvoiceInformationArray	8	0.00h	62.50%
ShortMessageSenderBean	send	2	0.00h	50.00%
XdslActivationManagerBean	activateBatchjobSingle	2	0.00h	50.00%
TelsisLKSystemBean	getProductCodes	61	0.00h	31.15%
TkanalApplicationBean	getAllSubscribedProducts	12	0.00h	25.00%
CustomerServiceBean	getAddressInformation	25	0.00h	16.00%
K2Bean	getUsersByKof2AccountID	45	0.00h	6.67%
K2ServiceBean	getUsers	45	0.00h	6.67%
AALUseCaseBean	sendTekniskK2	24	0.19h	4.17%
AccountServiceBean	getTransactionsForAccount	115	0.02h	3.48%
	getTransactionsForAccount2	115	0.02h	3.48%
NitraBean	getInterfaceDataBras	75	0.02h	1.33%
NitraBeanImpl	createInterfaceDataBrasOutput	75	0.00h	1.33%
K2Bean	reopenCustomer	84	0.05h	1.19%
K2ServiceBean	reopenCustomer	84	0.05h	1.19%
K2Bean	closeCustomer	200	0.14h	1.00%
K2ServiceBean	closeCustomer	200	0.14h	1.00%

Excel : Load Balancing

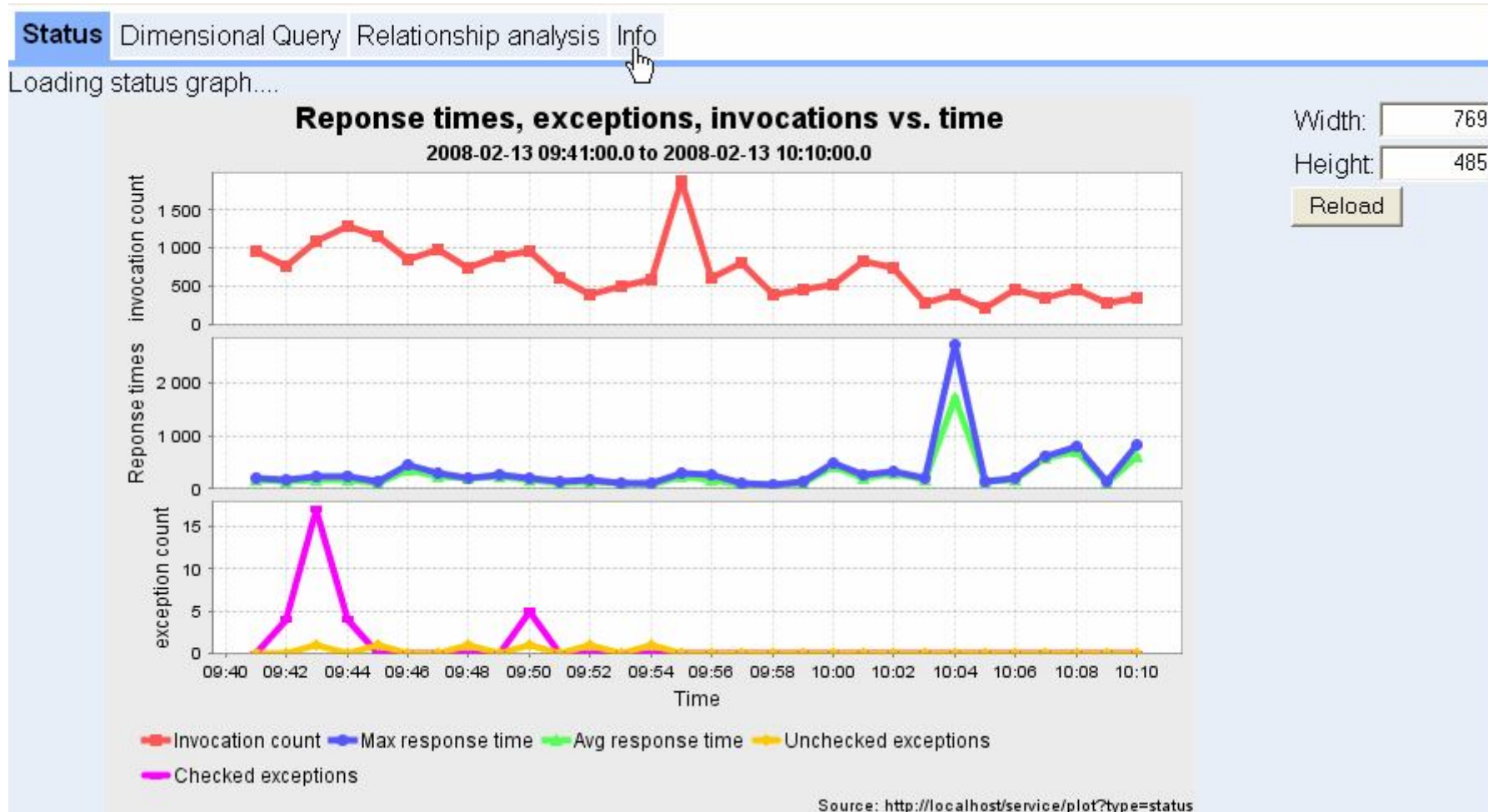
Metro2Cluster



appcluster



GWT based prototype GUI



Current state of project

- Release 1.0 available on February 15, 2008
- Deployed to production within next two weeks
- Collection and storage of observations – complete
- Processing of collected data requires manual operations



Future possibilities

- User friendly GUI, which does not require detailed knowledge of data model
- Detect abnormal behaviour through application of statistical methods
- Distribution of alerts via SMS, email etc.
- User friendly and domain specific analysis and reports
 - Most OLAP/BI tools require detailed knowledge
- Integration with source repository (Fish eye)
- Analyse SQL statements and look at usage trends