Model-Based Performance Management Techniques for Modern Applications

Geoffrey Vona

Development Manager
Quest Software
http://www.quest.com

TS-1591

Goal of This Talk

What Are You Going to Get Out of This?

Understand how model-based performance management can help you manage and understand your application

Or, a one hour nap. Your choice. Enjoy!

Model-Based Performance Management Techniques

- Definitions: monitoring and performance management
- Introduction: what is a model, and why is it helpful?
- History of monitoring and performance management
  - How applications are evolving
- Problems that break traditional monitoring:
  - Transactional flow data
  - Virtualized environments
- How models work
- Examples of models
- Summary and conclusions

Monitoring and Performance Management

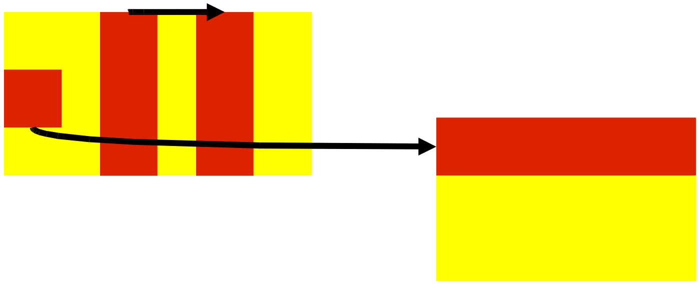
- Monitoring
  - Is it running?
  - Is it fast enough? (service levels)
  - Less data on all the parts
  - How much does it cost (resource consumption)?
  - Operations, production, reporting, management
- Performance management
  - Keeping it running
  - Making it run faster (or fast enough)
  - More data on fewer parts
    - Evolving to be broader
  - Design, capacity planning, profiling, diagnosis

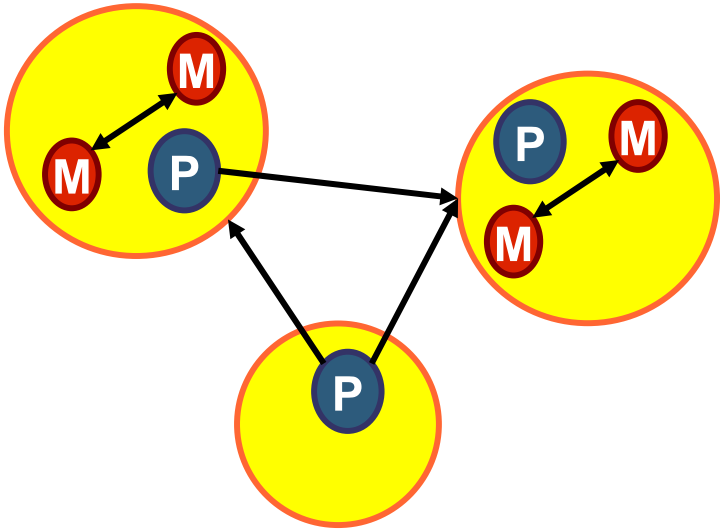
What Is a Model, and Why Is it Helpful?

- A model is a specific way of organizing data gathered about a system
  - Application of well-known object-oriented principals to the monitoring domain
- Model-based performance management involves turning raw collected data into a model of the underlying system
  - Should look like the picture an application owner would draw
  - Separates the context of the data from the data itself
  - Allows the same data to be used in different ways by different models
  - Allows different users to have different views on the data
- Model-based performance management helps
  - Reduce false alerts
  - Speed diagnosis
  - Uncover trends

How Does a Model Differ From Traditional Monitoring

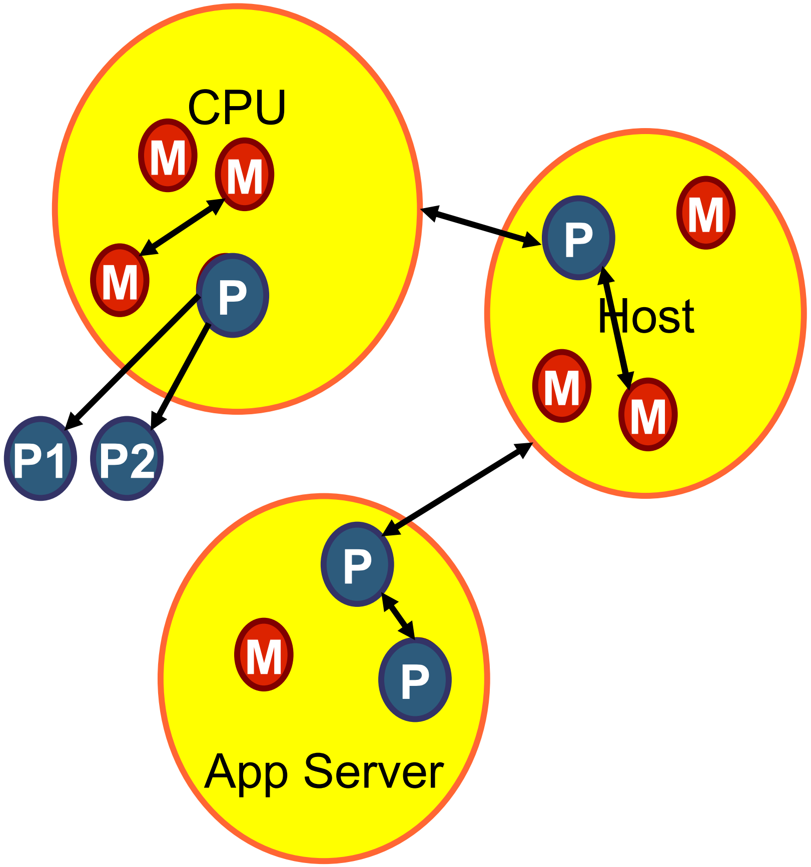
- Traditional monitoring organizes data by point of collection, or by metric type
  - Relational database tables
  - Correlation inside the table
  - Correlation using keys
  - Difficult to correlate outside the table
  - Cannot differentiate properties and metrics
- Model-based monitoring organizes data by monitored resource
  - Represents what is being monitored
  - Correlation is implicit in the location of the data





How Does a Model Work?

- Groups data into an object of a particular type
- Differentiates between metrics and properties
  - Metric: time-series data
  - Property: attribute of system that doesn’t change frequently
- Tracks changes to properties
- Uses properties and context to identify relationships with other objects
  - Including dynamic dependency mapping
- Allows correlation
  - Metrics to metrics
  - Metrics to properties
  - Properties to properties



Why Do I Care About Models?

- If you do performance management or monitoring for your applications, you need better data
    - Application scale and complexity is increasing
    - Current state of the art is presenting more uncorrelated data
        - Uncorrelated data can help sometimes, but some problems cannot be solved
    - Gathered data set changes at run time
- With the basics of models in mind, let’s examine the history of monitoring and evolving application complexity
    - Make the case for a new approach

History of Monitoring

- Phase One: Availability
  - Is it running?
- Phase Two: Proprietary performance data
  - Why isn’t it running?
  - Monitoring vendors provide performance data
  - Platform vendors provide performance data
- Phase Three: Standardization
  - JSR 77, Java™ Management Extensions (JMX™)

History of Monitoring Part II

- Phase One: Your environment was a simple network map showing everything
  - Available or not available
- Phase Two: Add more data to the network map
- Phase Three: The data often contains relationship and property information
  - JSR 77 has a rich object hierarchy and can represent properties as well as metrics
  - Domain-specific models
  - But what about cross-domain data?

How Applications Are Evolving

- Huge changes in the last 10 years
  - Your application is not just yours
    - Integrations, other groups
  - Your part of the application is not all your code
    - Frameworks, open source
- These changes have made old school techniques obsolete
  - Not the good old school

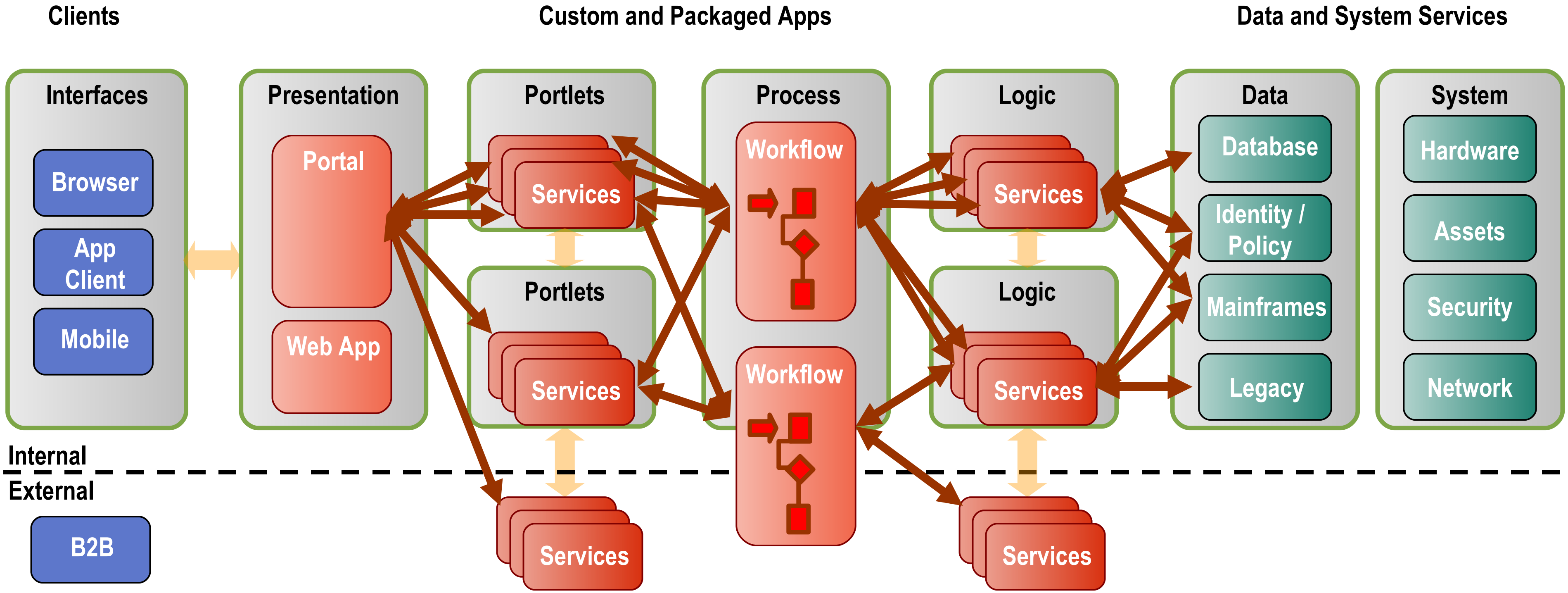
Application 1996

- In the early days, getting your application on the Internet was cool enough
  - Servlet using JDBC™ software to call a database
- Infrastructure was fairly simple
  - <5 systems, including 1 database
  - Isolated: low cost, therefore dedicated hardware made sense
  - Isolated: single group owned the whole thing
  - Transaction volumes were small
    - Didn't seem like it at the time

Application 2006

- Applications have new complexities along all possible axes
  - Huge infrastructure that is shared with other applications
    - Or, ASP model—somebody else provides it
    - Or, grid—infrastructure is flexible
  - Infrastructure is shared with other groups
    - Multiple interests are being served: systems group, database group, etc.
  - Incorporation of legacy systems
  - Frameworks, open source
  - Specialty servers
  - Platforms e.g. workflow servers and ERPs built on top of the Java Platform, Enterprise Edition (Java EE)
  - SOA

Application 2006: Complexity Rules



Clients

Custom and Packaged Apps

Data and System Services

Interfaces

Browser

App Client

Mobile

Presentation

Portal

Web App

Portlets

Services

Portlets

Services

Process

Workflow

Workflow

Logic

Services

Logic

Services

Data

Database

Identity / Policy

Mainframes

Legacy

System

Hardware

Assets

Security

Network

Internal

External

B2B

Services

Services

Organizing Application Complexity

- Vertical complexity
  - Add complexity inside a single piece
  - Tiers, isolation layers, frameworks
- Horizontal complexity
  - Add pieces and paths between them
  - Clustering, dynamic deployment, virtual environments
- Heterogeneity
  - Adding complexity by adding variables on the pieces
  - Server types, legacy systems, changing deployments, Web Services

Application Complexity: So What?

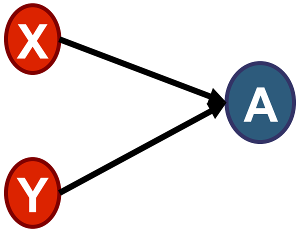
- A big problem if you need to manage application performance
  - How are you going to get data from all the domains?
    - Probably a mixture of tools you buy, tools you download and tools you build yourself?
  - How are you going to correlate that data across domains?
    - Excel?
  - What if your domain changes—do you lose your ability to manage the domain?
    - Sometimes the domain changes are mandated
- The relationships must be preserved!
  - Enter model-based monitoring

Problems That Break Traditional Monitoring: Transaction Flow Data

- Transaction flow data is gathered using instrumentation that can monitor an in-flow or out-flow
  - Call tree
  - Each node has metrics on the performance of a “method” or “tier”
  - Each node also has relationships with other nodes
    - App server X calls app server Y calls database Z
- Absolutely critical for SOA and Web Services
  - What are the dependencies?

The Transaction Flow Problem

- Web server X, Y call app server A
  - X has a caching error that causes too many calls to the app server tier
    - X->A: 26000
    - Y->A: 23



  - If the relationship is not preserved, all you have is call count for A
  - Similar: if Y is misconfigured so it never calls A

Problems That Break Traditional Monitoring: Virtualized Environments

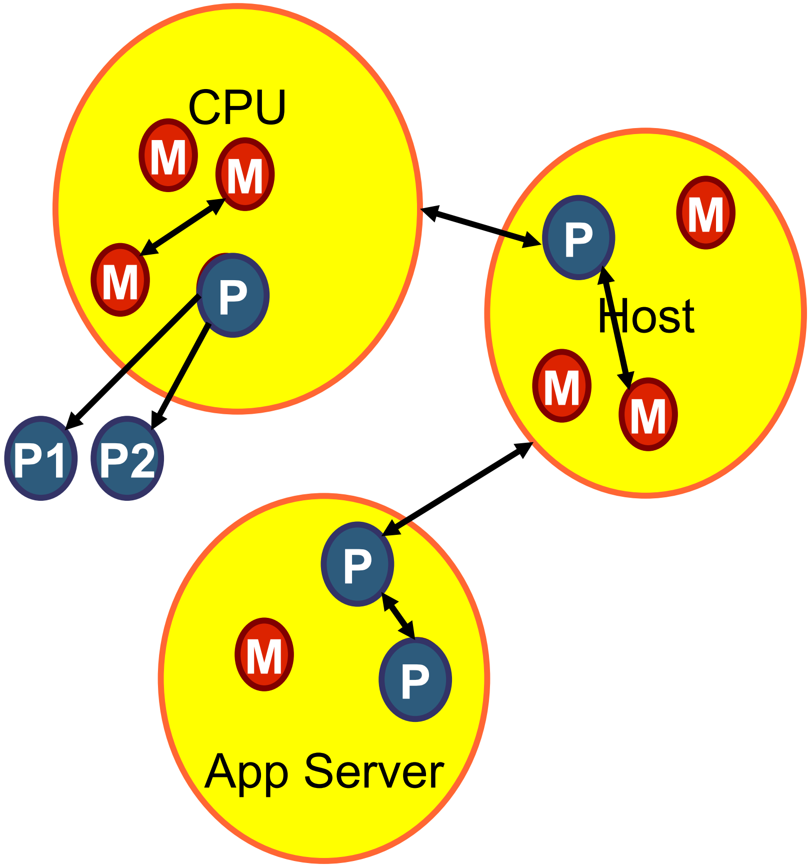
- Host monitoring used to be easy
  - A host was always a host
  - 1:1 mapping between logical and physical host
- Today’s world is much more complex
  - A host may be a logical host representing an active-passive cluster
  - A host may be one of a cluster of hosts arbitrarily grouped together
  - A host may be a virtual host running on a physical host
  - A host may change its IP address or domain dynamically

Problems That Break Traditional Monitoring: Virtualized Environments

- Two problems:
  - The relationship between physical and logical must be preserved, or the host data becomes meaningless
    - The Wichita lab is shutting down all servers for a scheduled power outage. They have provided a list of physical boxes that will be shut down. Will I be affected?
    - A host goes down, and my active-passive cluster switches to one of the passive nodes. Do I get an alarm because my original host was tied to my app server data?
  - The property information must be tracked—changes could require groupings to change
    - Example: Hosts are grouped by IP address, IP address changes

How Do Models Work?

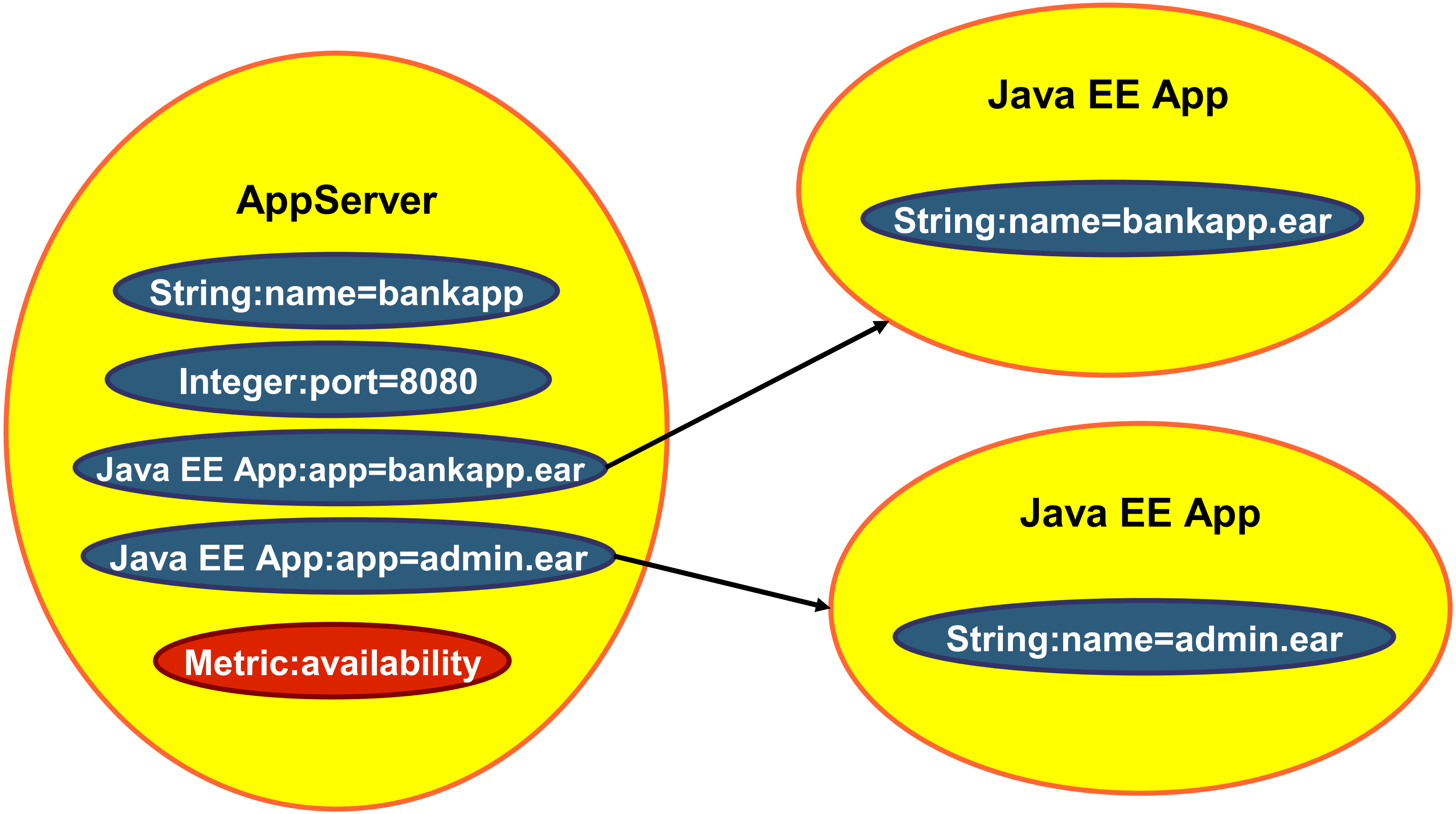
- Remember that we said that models:
  - Group data into objects
  - Create properties from metrics
  - Track changes
  - Preserve relationships
  - Enable correlation



Attributes of a Model

- Models are made up of objects that have:
    - Properties
    - Relationships
    - Metrics
    - Alarms and changes
- A property can have multiple items
    - Single entry or list
- In terms of the actual implementation, everything is a property
    - Relationships, metrics, alarms, changes are typed specializations of a property

Model Capabilities: Properties and Relationships



AppServer

String:name=bankapp

Integer:port=8080

Java EE App:app=bankapp.ear

Java EE App:app=admin.ear

Metric:availability

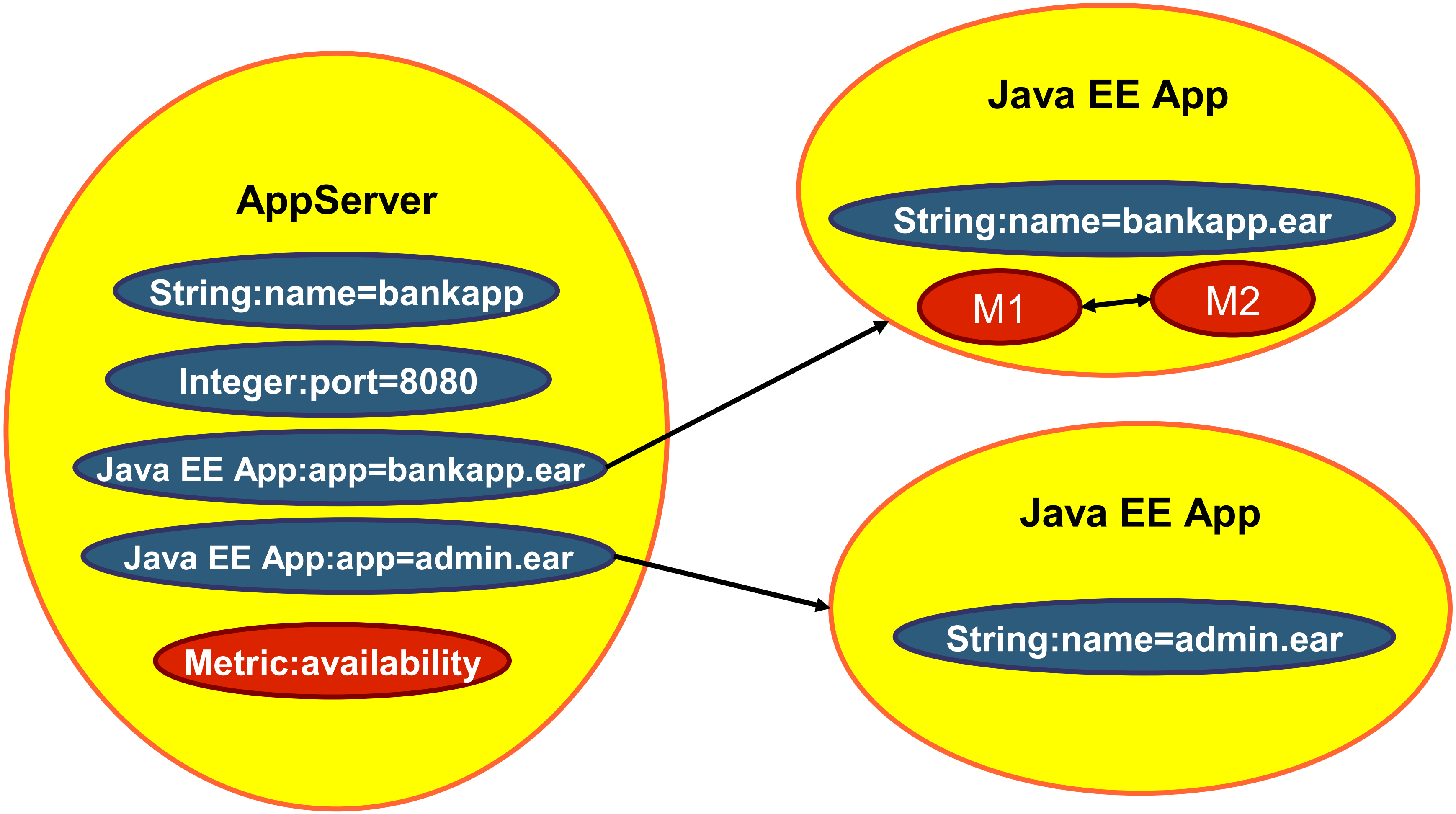
Java EE App

String:name=bankapp.ear

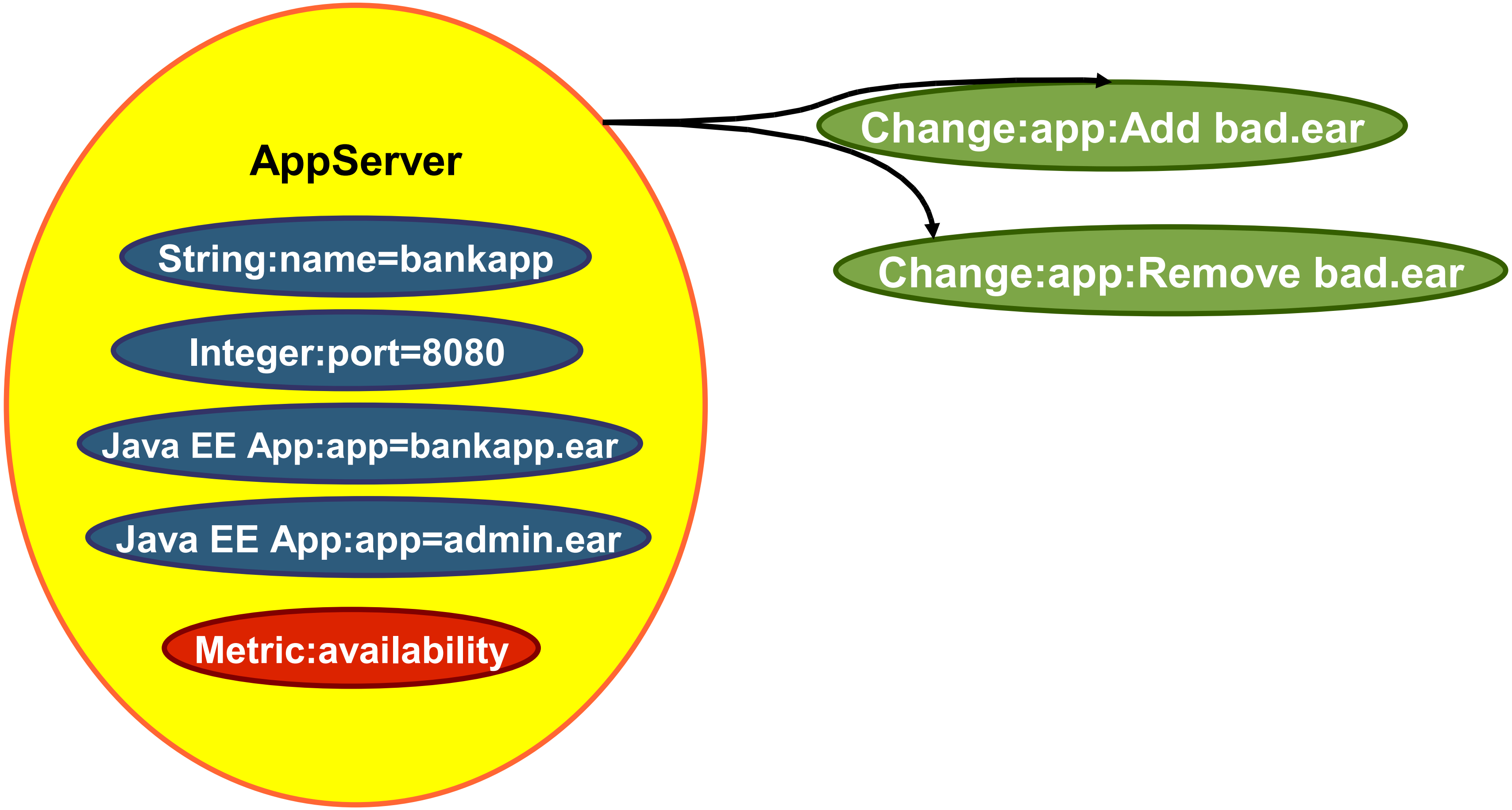
Java EE App

String:name=admin.ear

Model Capabilities: Implicit Data Correlation



Model Capabilities: Tracking Property Changes



AppServer

String:name=bankapp

Integer:port=8080

Java EE App:app=bankapp.ear

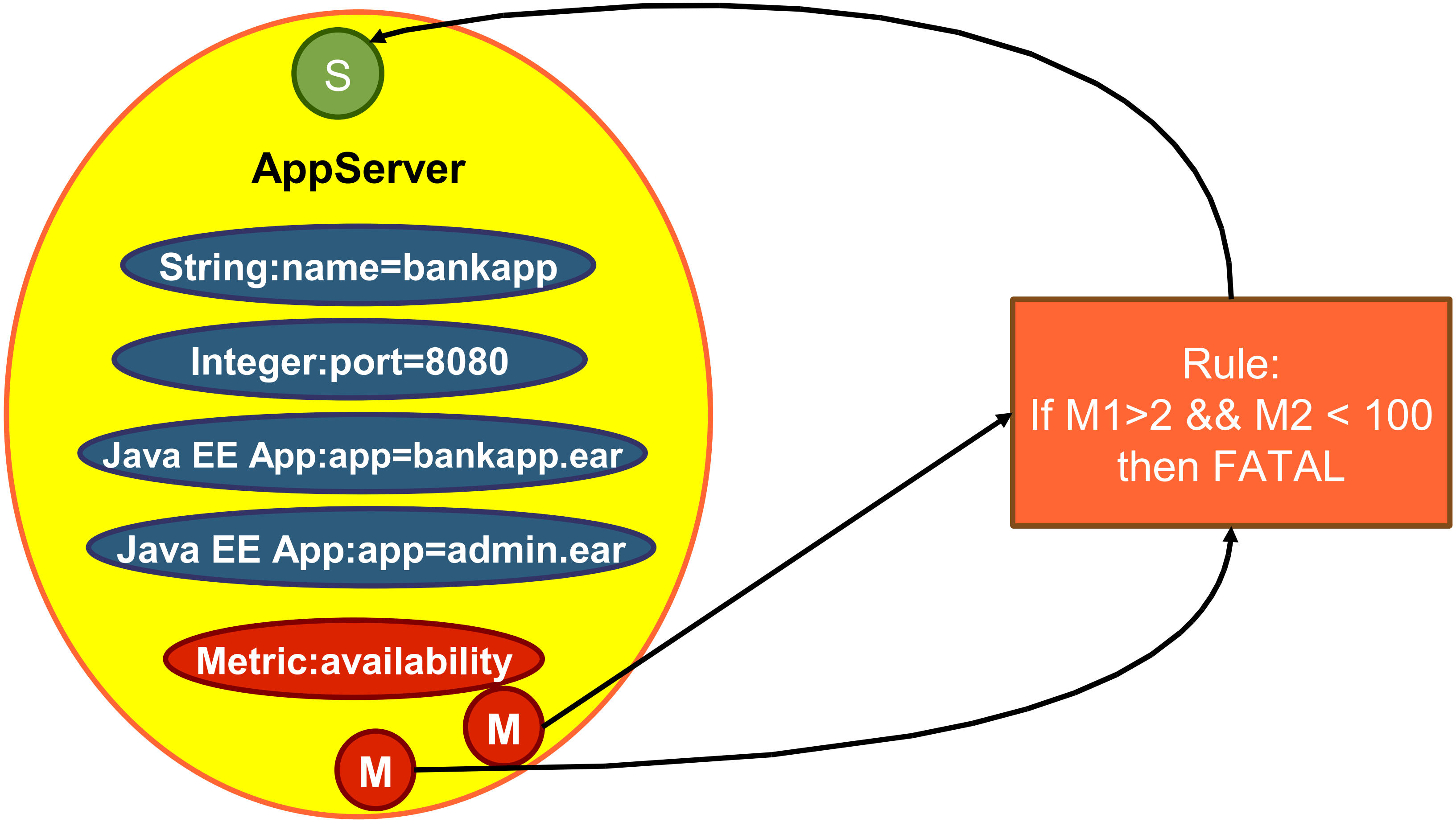
Java EE App:app=admin.ear

Metric:availability

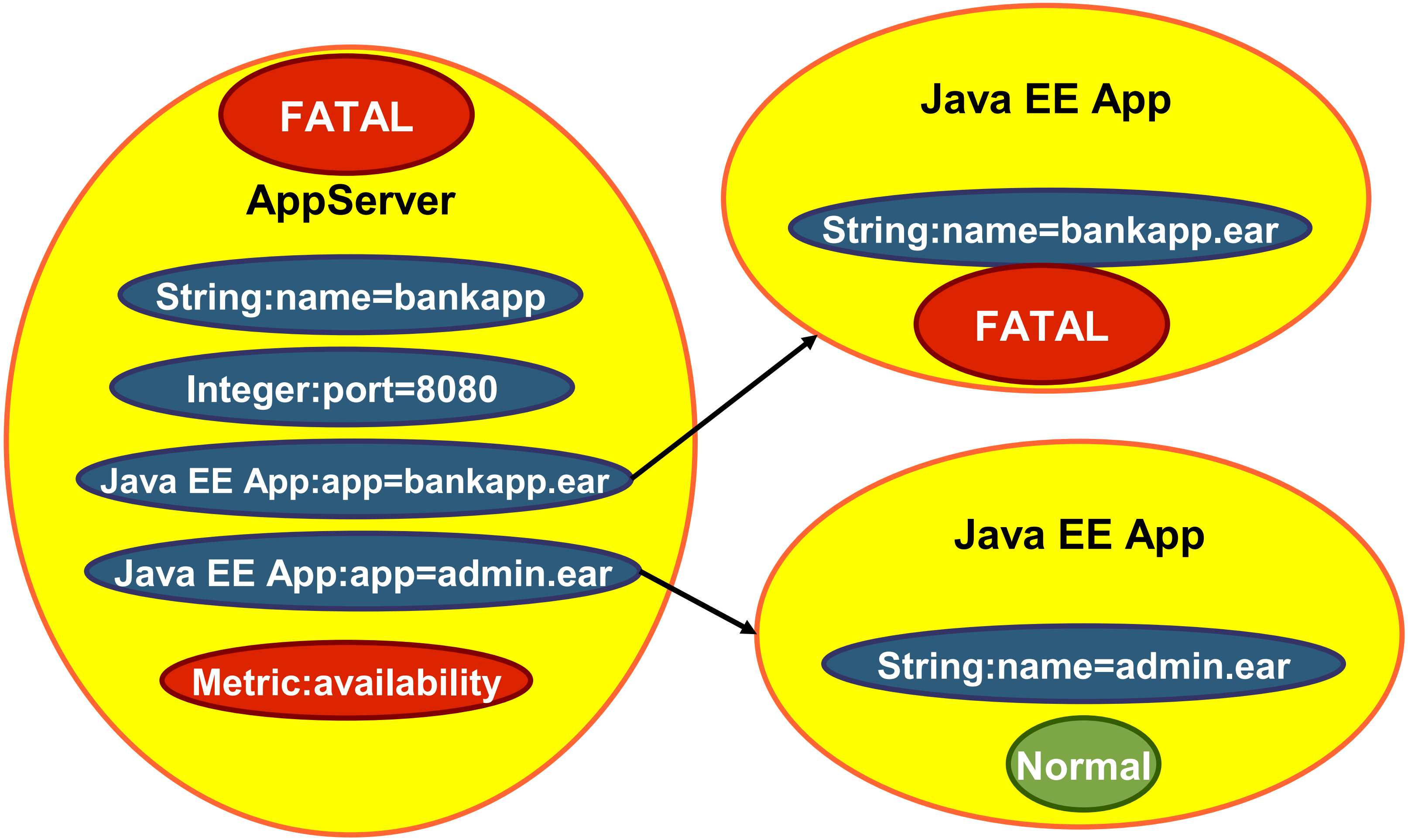
Change:app:Add bad.ear

Change:app:Remove bad.ear

Model Capabilities: State Annotations



Model Capabilities: State Propagation

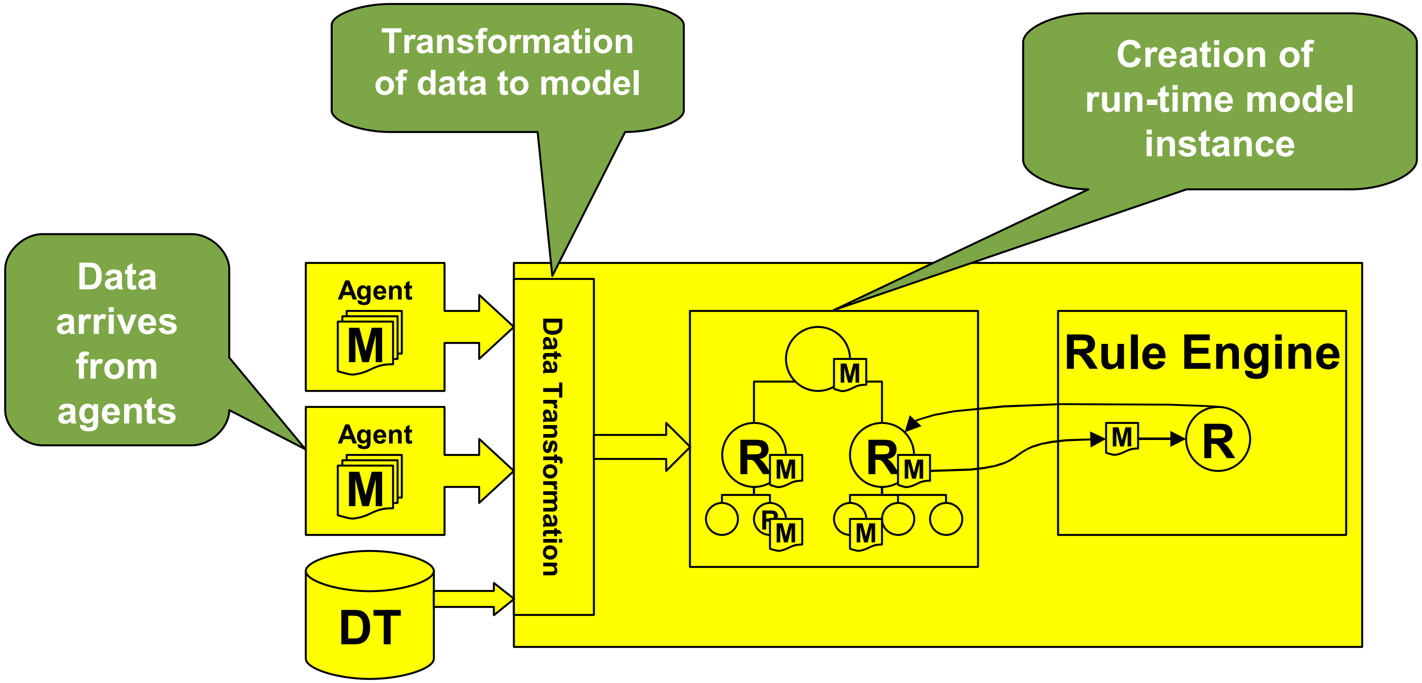


Runtime Behaviour of Models

- Models can be created and updated based on data available today
    - Combination of raw data and collection context
- What is required is a transformation of the data from metrics to model objects, properties, property changes and metrics
    - Could be done as part of data post-processing
    - Could be done dynamically
        - Benefits include responding to change, intelligent alerting

Dynamic Data Transformation Architecture

- Configure with data transformation
- Transform data as it arrives



Benefits of Dynamic Data Transformation

- Model can respond to real changes in the environment
- Model can change as new entities come online
  - Or as new types of data collection are enabled
- Changes can be tracked as property changes
- No one-time calculation
- Data transformation definition can change
  - Create multiple models from the same data

More Benefits of Dynamic Data Transformation

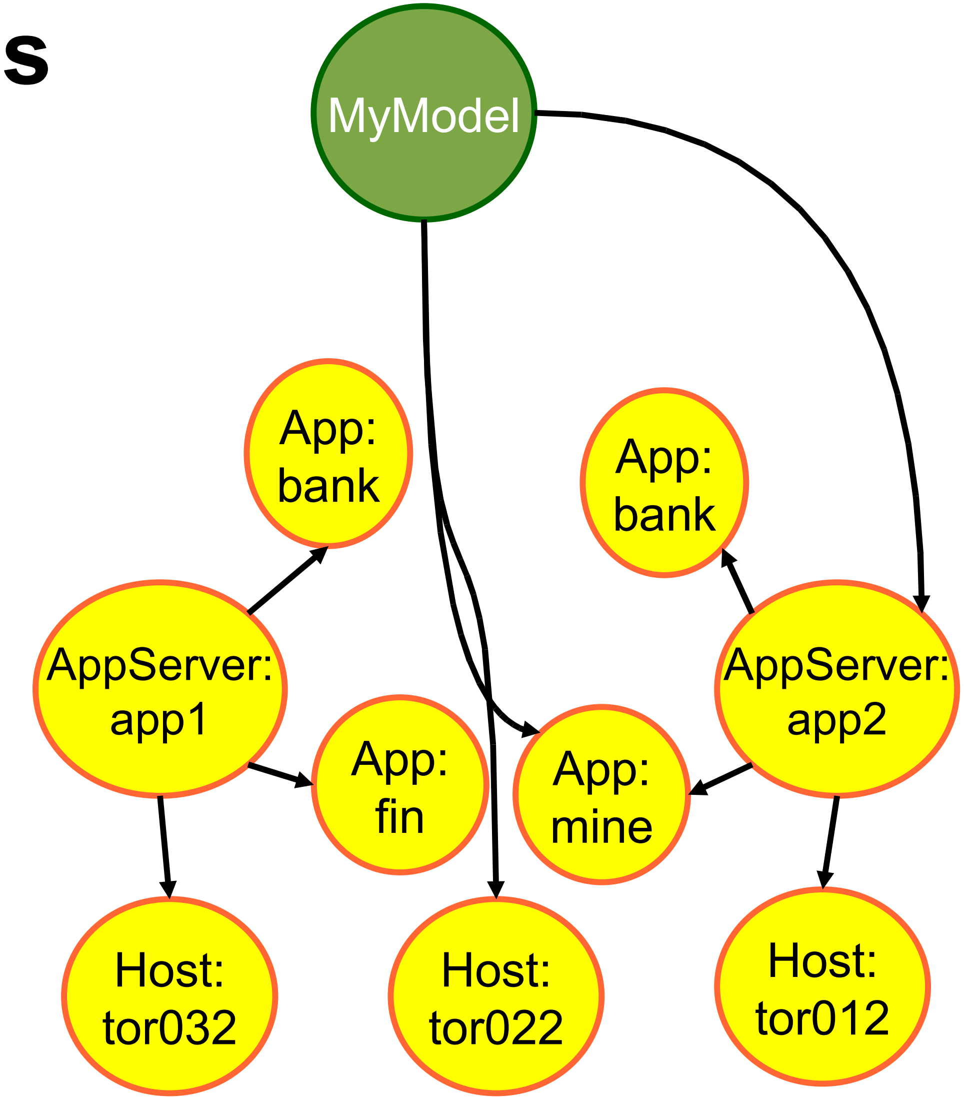
- Result of transformation is a common form for the data
  - Enables correlation and post-processing
- When data is transformed and placed in a model, the original collection context is no longer important
  - Data is not “stamped” by where it was collected
  - Enables remote/touchless collection
  - Enables cluster collection
- Data from different collectors can rendezvous in the same object

How Models Enable Intelligent Alerting

- As mentioned earlier, by preserving all relationships, models enable state propagation
  - Alerts are associated with the originating model object
  - Functions can be written to propagate state in interesting ways
    - Creation of Service Level Agreements (SLAs) like “AppServerCluster not available if more than 2 out of 5 nodes are down”
    - Difficult to do without preserving the relationships.
    - Very important for reporting and chargeback

Aggregate Models

- Models can be combined in arbitrary ways
  - Application groupings
  - Logical groupings
  - Organizational groupings
  - Geographical groupings
- Same features exist for aggregate models
  - State is propagated
  - Property changes are tracked
  - Metrics can be associated and correlated
- Now we enter a very powerful domain!



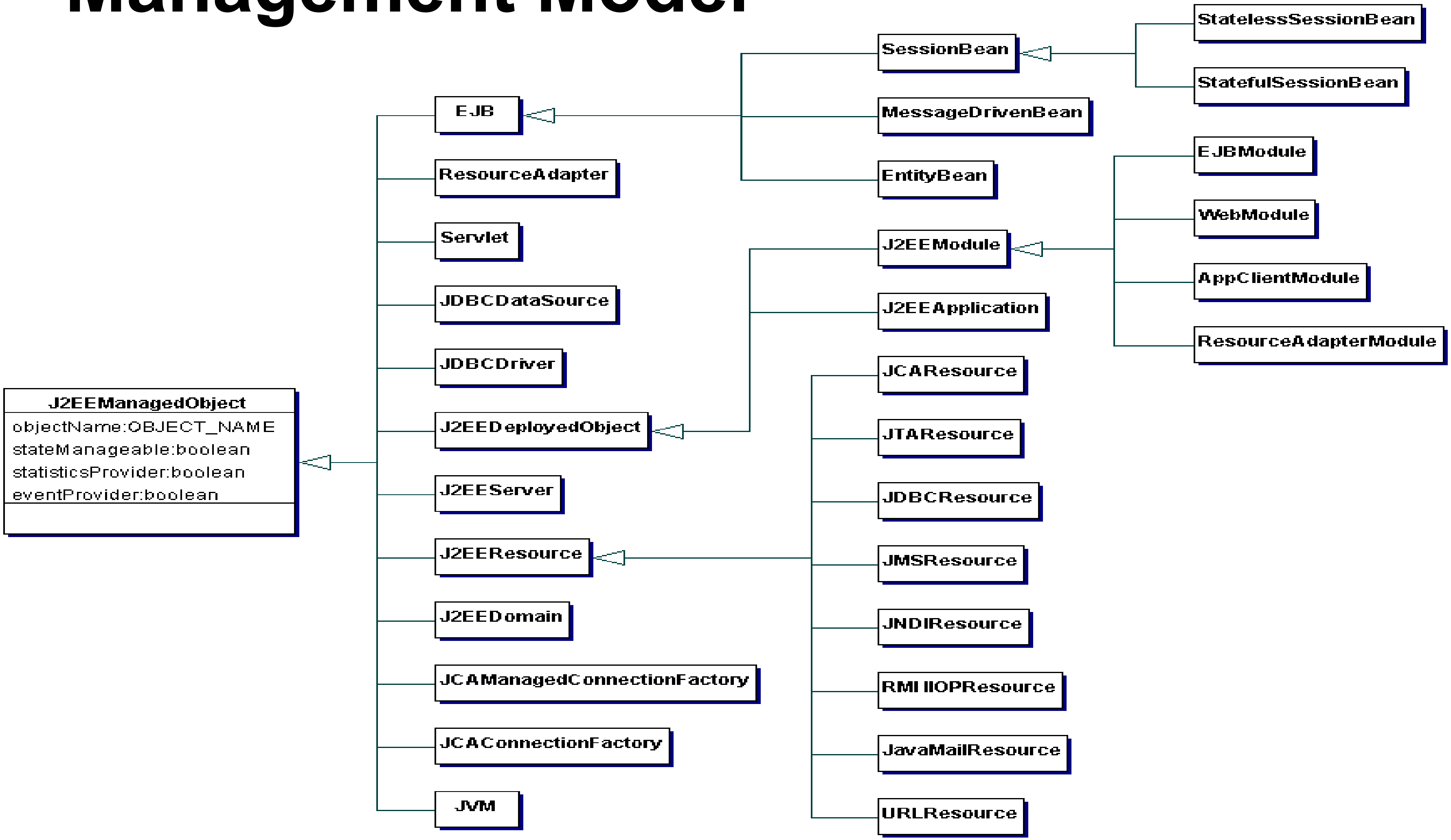
Example of an Aggregate Model

- App server farm is provided for application developers at BigBucksBank
- Multiple applications hosted per app server
- User can create an aggregate model that selects only the applications she cares about
  - Groups them in a way that makes sense
  - Define custom ways of propagating state
- State is only propagated for the things she cares about
  - No looking at someone else’s problems all the time ☺

Do Models Exist Today?

- Models do exist today
- JSR 77 does a good job of defining a performance management model for the Java EE platform
- Most application servers have JMX API MBeans that are a model
  - Unfortunately, most collection technologies jettison the object relationships
- CIM defines models for most domains
  - Although this definition is relatively shallow
  - Not frequently used

JSR 77 Java EE Performance Management Model



Do We Need More Than Domain Models?

- Performance management is increasingly about bridging technology silos
    - Multiple app servers, web servers, database
- Domain models are great, but something needs to bring it all together
- Need to be able to create models for domains that are flat
- Need to be able to create custom aggregate models to represent a true application owner’s slice on the systems

Summary

- Application complexity requires a new approach to performance management and monitoring
- A model-based approach uses raw data to create objects that represent parts of an application
- Models have properties, relationships, metrics and state
- A model-based approach
  - Preserves context
  - Enables data correlation
  - Structures system state
  - Allows model elements to be rearranged in arbitrary ways

Model-Based Performance Management Techniques for Modern Applications

Geoffrey Vona

Development Manager
Quest Software
http://www.quest.com

TS-1591