CSCI 5258 Foundations of Software Engineering

# Software Architecture

# A Model Driven View

Dong Chen

Email: zaknova@gmail.com

University of Colorado, Boulder

# Outline

- Software Architecture
- Case Study: Model Driven Architecture
- Application Case Study: ICDE
- Reference

# Software Architecture

- Definition
- Understanding
- Nonfunctional requirements
- Modeling of Architecture and Baseline
- Software Architects
- Different software architectures

# Software Architecture Definition

## Components and their interactions:

- Architecture is defined by the recommended practice as the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.[ANSI/IEEE Std 1471-2000, Recommended Practice for Architectural Description of Software-Intensive Systems]

## Abstraction:

- The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. [L.Bass, P.Clements, R.Kazman, Software Architecture in Practice (2nd edition), Addison-Wesley 2003]

# Definition Cont…

**Scalability, distribution, etc.**

- [Software architecture goes] beyond the algorithms and data structures of the computation; designing and specifying the overall system structure emerges as a new kind of problem. Structural issues include gross organization and global control structure; protocols for communication, synchronization, and data access; assignment of functionality to design elements; physical distribution; composition of design elements; scaling and performance; and selection among design alternatives. [D. Garlan, M. Shaw, An Introduction to Software Architecture, Advances in Software Engineering and Knowledge Engineering, Volume I, World Scientific, 1993]

# Understanding Architecture

- Defines structures

  a. Partitioning into components, modules, or any units.

  b. Minimizing dependencies and loosely coupled

- Specifies components communication

  Pattern specifies certain component communications

  e.g. client-server pattern: providing mechanisms for connection establishment, error handling, server security, etc.

# Nonfunctional Requirements

- Three nonfunctional requirements:

  a. Technical constraints: specifying technologies used

  b. Business constraints: specifying business design options

  c. Quality attributes: scalability, availability, portability, etc.

- Abstraction:

  a. Informal description of systems (marketecture)

  b. Hierarchical decomposition

- Architecture views

  Logic view: expressing logic using class diagrams or others

  Process view: describing concurrency and communications

  Physical view: mapping to the application hardware

  Development view: internal organization of software components

# Modeling of Software Architecture and Baseline

- Modeling

  Using use case and class diagrams, dynamically using sequence , collaboration , state charts , and activity diagrams

- Baseline

  **Requirements:** critical use cases, system level quality objectives, priority relationships among features and qualities.

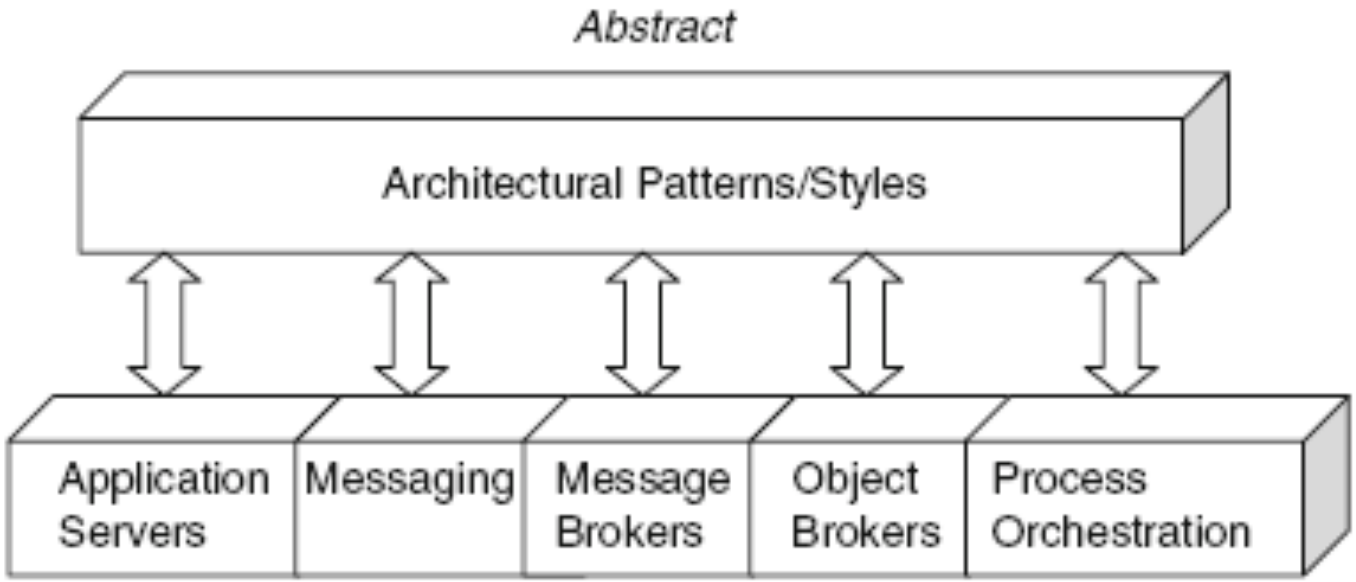
  **Design:** names, attributes, structures, behavior, groupings, and relationships of various classes and components.

  **Implementation:** source component inventory and bill of materials of all primitive components.

  **Deployment:** executable components, risk associated with the system components.

# Software architects

- Multi-skilled in software engineering, technology, management and communications
- Mapping the abstract patterns to specific implementations to meet the requirements
- Philippe Krutchen: The life of a software architect is a long (and sometimes painful) succession of sub-optimal decisions made partly in the dark



# Different software architectures

- Middleware Architecture
- Object-Oriented Architecture
- Resource-Oriented Architecture
- Service-Oriented Architecture
- Aspect-Oriented Architecture
- **Model-Driven Architecture**
- …etc.

# Case Study: Model Driven Architecture

- MDA
- Principles of MDA
- Model Transformation
- MOF in MDA
- Why MDA?
- MDA and SOA

# Model Driven Architecture (MDA)

- Abstraction levels in software industry in past five decades: From machine code to assembly language to 3GLs to object-oriented languages and now to models
- MDA: “an approach to IT system specification that separates the specification of functionality from the specification of the implementation” defined by OMG
- Simply a bunch of models and their transformations
- State of art Tools using MDA: AndroMDA, ArcStyler, Eclipse Modeling Framework

# Principles of MDA

Four principles underlie the OMG's view of MDA:

- Models expressed in a well-defined notation are a cornerstone to understanding systems for enterprise-scale solutions.
- The building of systems can be organized around a set of models by imposing a series of transformations between models, organized into an architectural framework of layers and transformations.
- A formal underpinning for describing models in a set of meta-models facilitates meaningful integration and transformation among models, and is the basis for automation through tools.
- Acceptance and broad adoption of this model-based approach requires industry standards to provide openness to consumers, and foster competition among vendors.

# Model transformations

- Computation Independent Model (CIM)

  Hide computational and implementation details.

  System's environment and requirement are emphasized

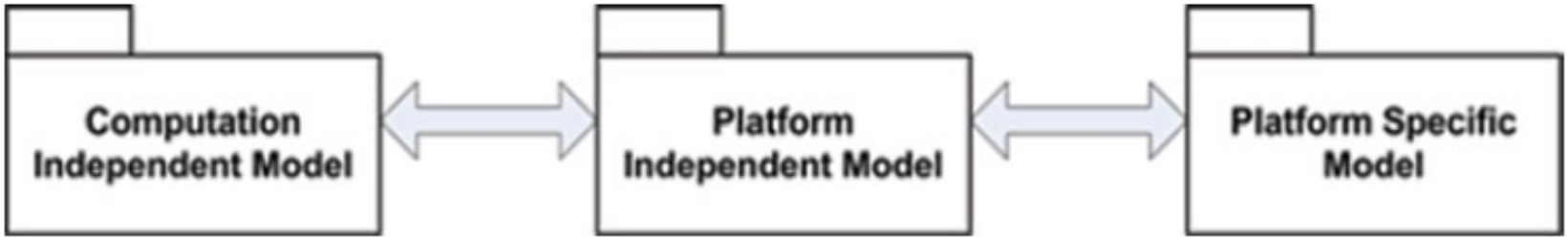
- Platform Independent Model (PIM)

  Transformed from CIM without platform specific information.

  Possess computational information for the application.

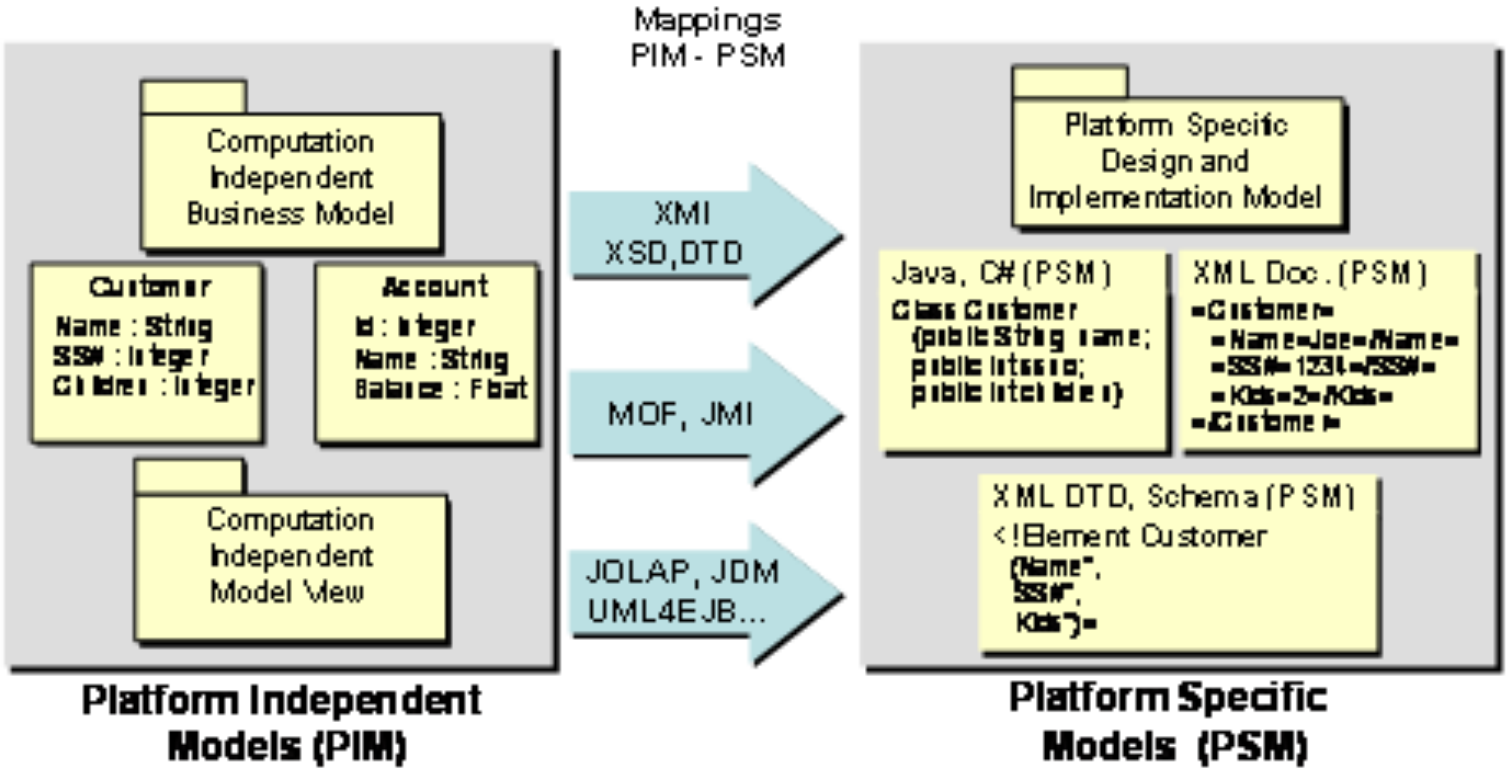
- Platform Specific Model (PSM)

  Transformed from PIM with details on specific platform implementation



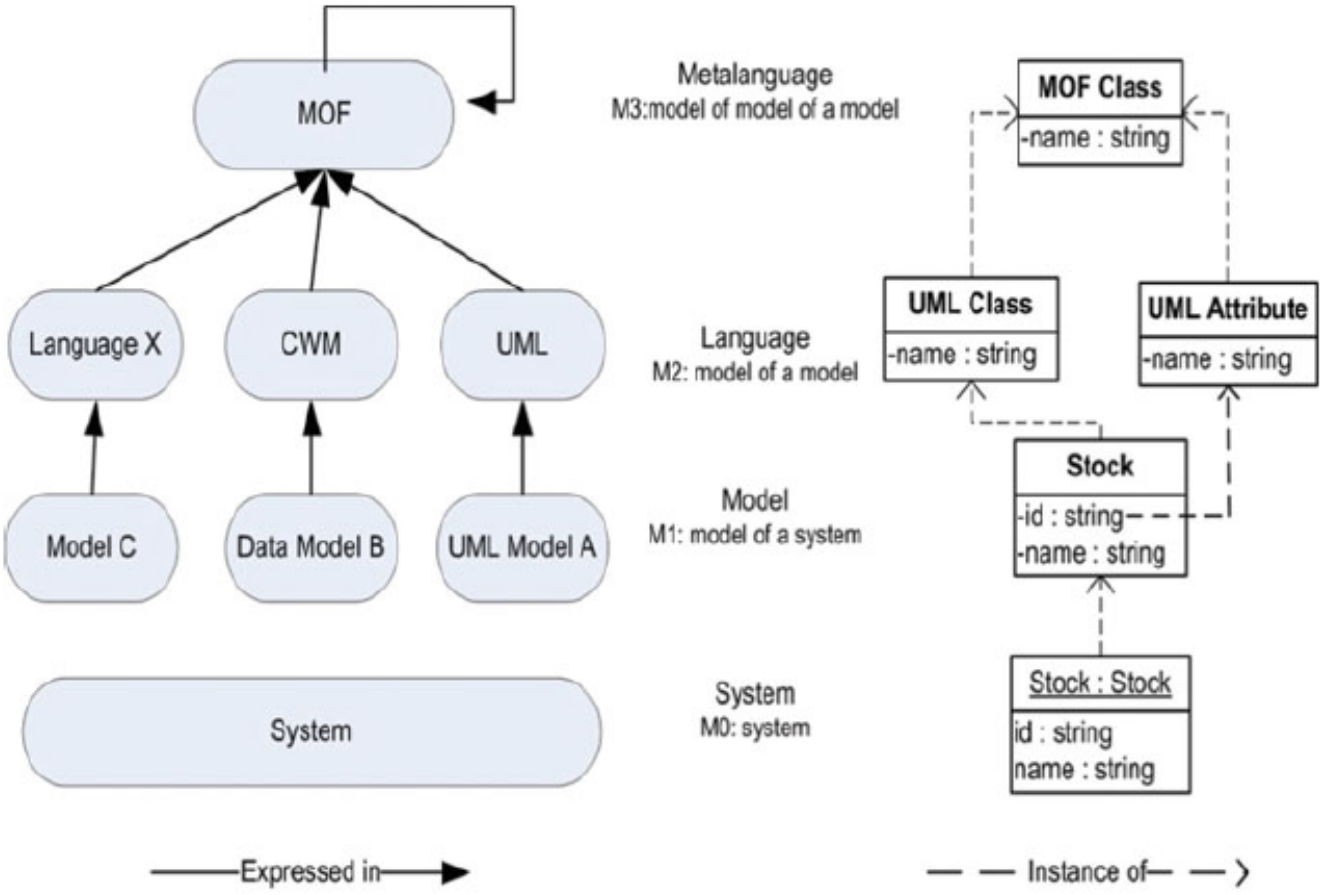
# Example

The following figure represents a Customer and Account. At this level of abstraction, the model describes important characteristics of the domain in terms of classes and their attributes, but does not describe any platform-specific choices about which technologies will be used to represent them. It also illustrates three specific mappings, or transformations, defined to create the PSMs, together with the standards used to express these mappings.



# MOF in MDA

- Meta-Object Facility (MOF): Create MODF representation of existing modeling languages (such as UML) to make them MDA compatible



# Why MDA?

- Portability:

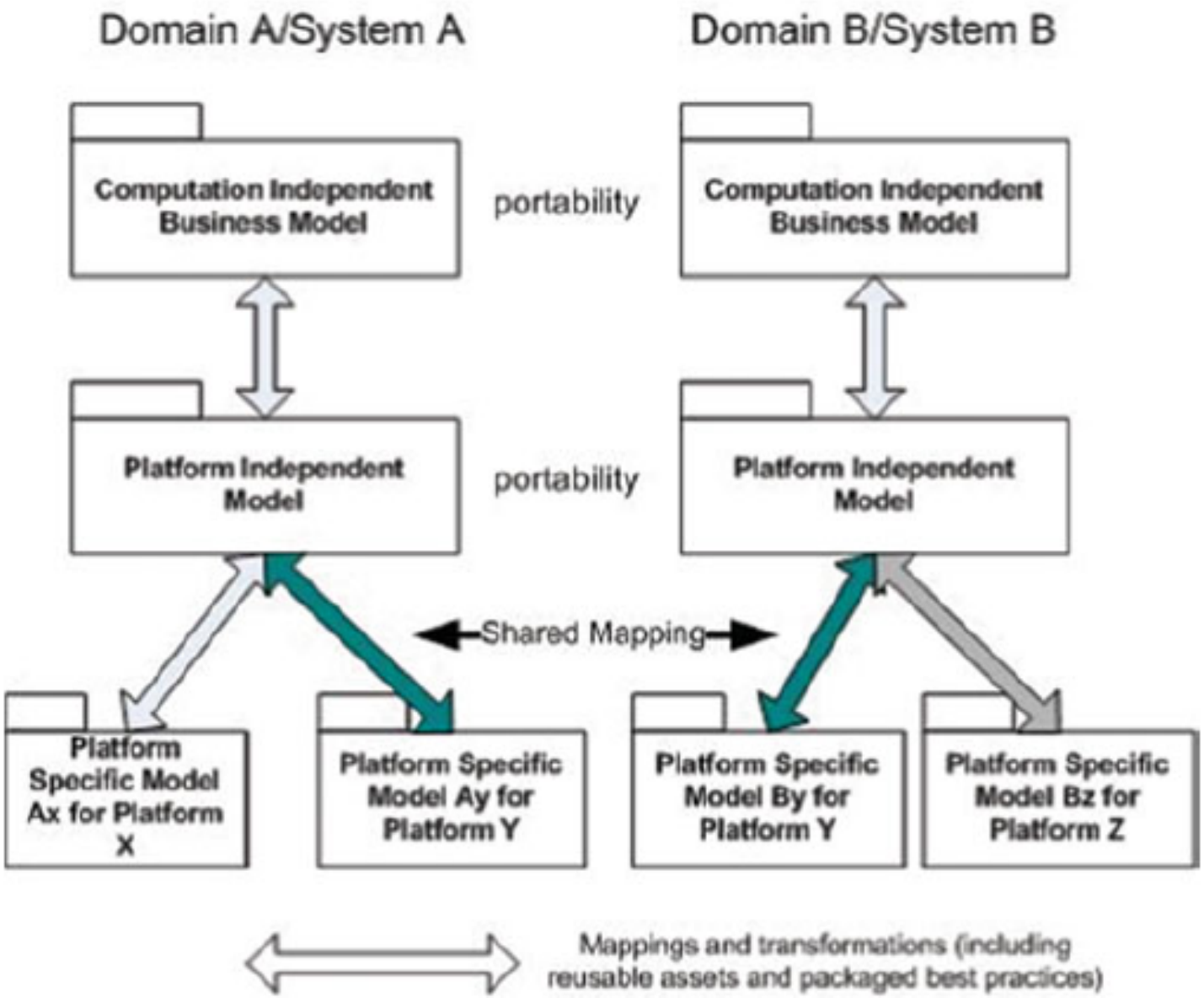
  a. High level models are decoupled with low level platform details.

  b. Do not need remodeling but transformation when underlying platform changes

  c. MOF makes models movable across different environments

- Reusability

  e.g. PIM is mapped to different PSMs for different platforms


Domain A/System A
Domain B/System B
Computation Independent Business Model
portability
Computation Independent Business Model
Platform Independent Model
portability
Platform Independent Model
Shared Mapping
Platform Specific Model Ax for Platform X
Platform Specific Model Ay for Platform Y
Platform Specific Model By for Platform Y
Platform Specific Model Bz for Platform Z
Mappings and transformations (including reusable assets and packaged best practices)

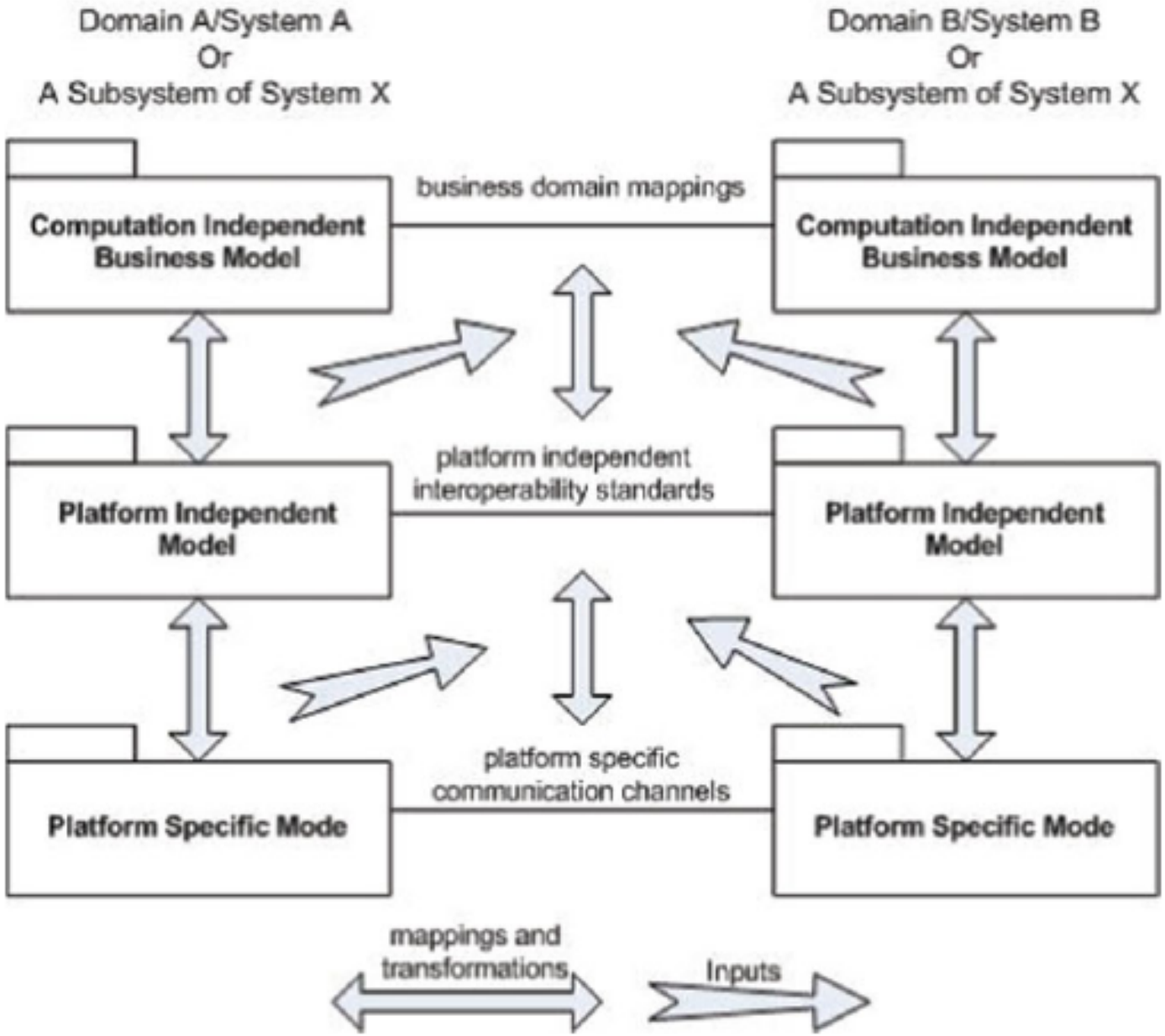
# Why MDA? Cont…

- Interoperability

a. horizontal model mapping and interactions.

e.g. two sets of CIM/PIM/PSM for the two systems. First explicit vertical transformation between high level models CIMs and PSMs can be analyzed. The cross platform model mappings can be mapped to detailed communication protocols or shared databases.

b. mapping a single high level model into multiple models across two or more platforms.


Domain A/System A
Or
A Subsystem of System X
Domain B/System B
Or
A Subsystem of System X
Computation Independent Business Model
business domain mappings
Computation Independent Business Model
Platform Independent Model
platform independent interoperability standards
Platform Independent Model
Platform Specific Mode
platform specific communication channels
Platform Specific Mode
mappings and transformations
Inputs

# MDA and SOA

- Difficulty of architecture design

  Systematically check whether the architecture models fulfill the requirements

- SOA (service-oriented architecture)

  a. Different perspective:

  SOA: communication protocols and architecture style perspective

  MDA: general semantic modeling perspective

  b. SOA uses communication protocols, pervasive services, etc, to bridge different systems.

  MDA applies transformation rules for the high level down to low level

# Application Case Study: ICDE

- ICDE System
- Extended Capacity Planning
- Reasons for choosing MDA
- MDA based Test Generator
- Test Results and Practical Merits

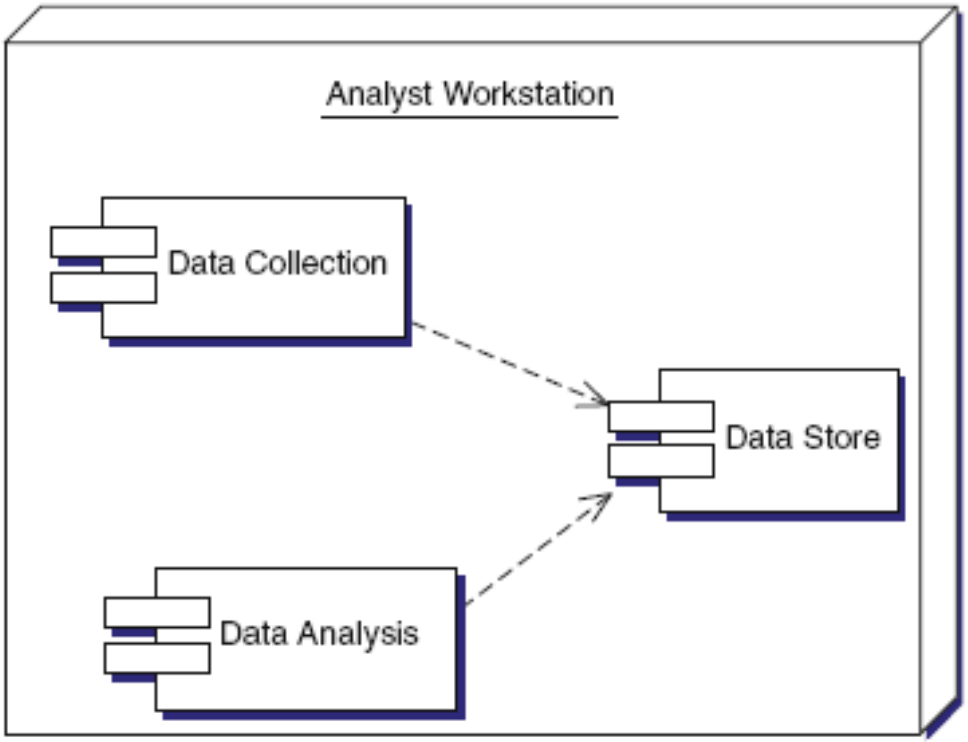
# Case Study on ICDE

- ICDE (Information Capture and Dissemination Environment) Initial objective: capture user actions use cases and offer intelligent helps

  a. Data Collection:  capture users' activities

  b. Data Store: database storage of event information

  c. Data analysis: analysis for the data store



# ICDE capacity planning

Promote the initial ICDE with network capability

- Different  domains and user installations use ICDE in different ways
- Different installations of ICDE on different hardware platforms
- Different application servers have different performance characteristics

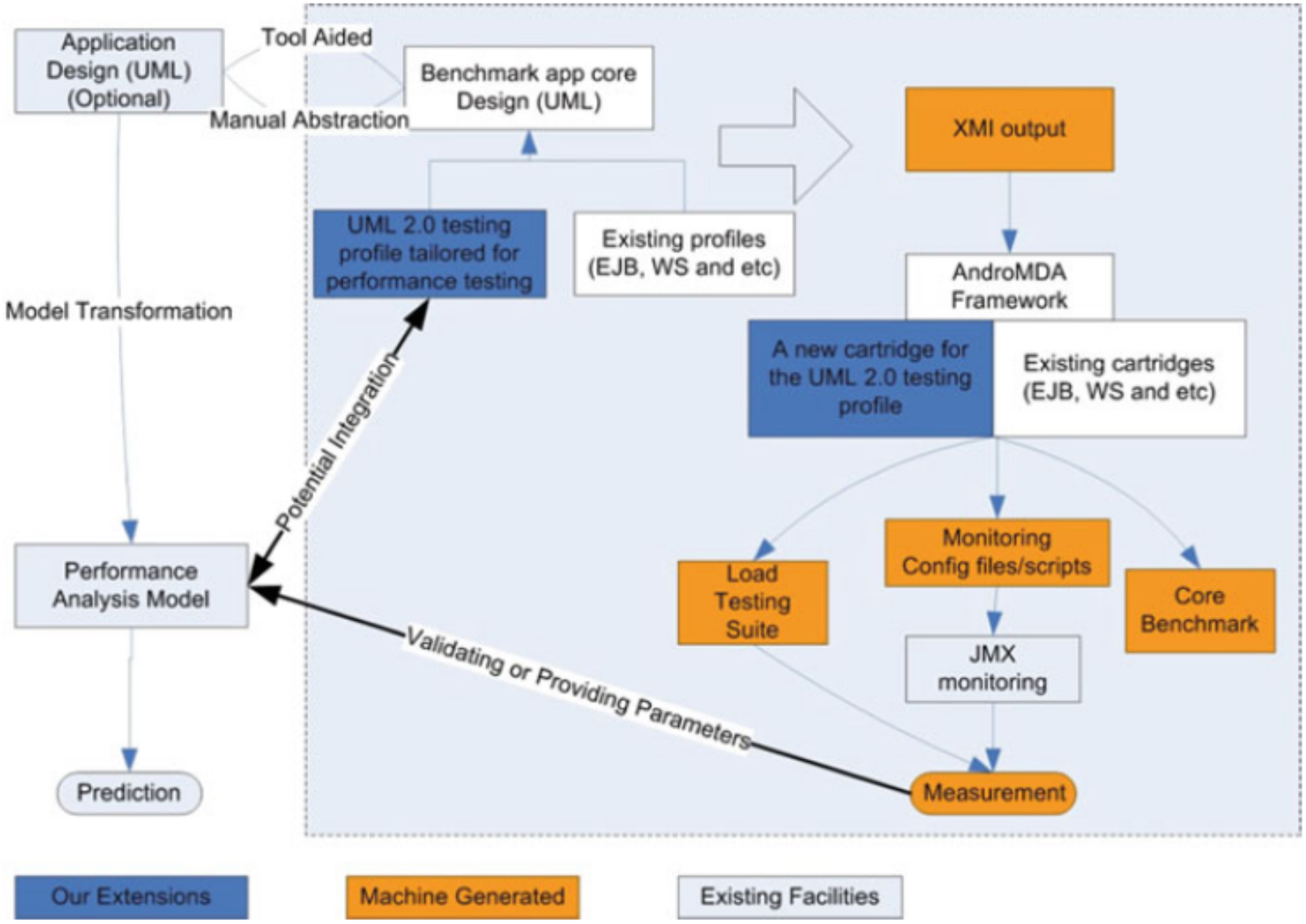
- Capacity planning is to execute a test load on specific platforms: then how to make test as efficient and painless as possible?
- Ans: applying MDA

# Reasons for choosing MDA

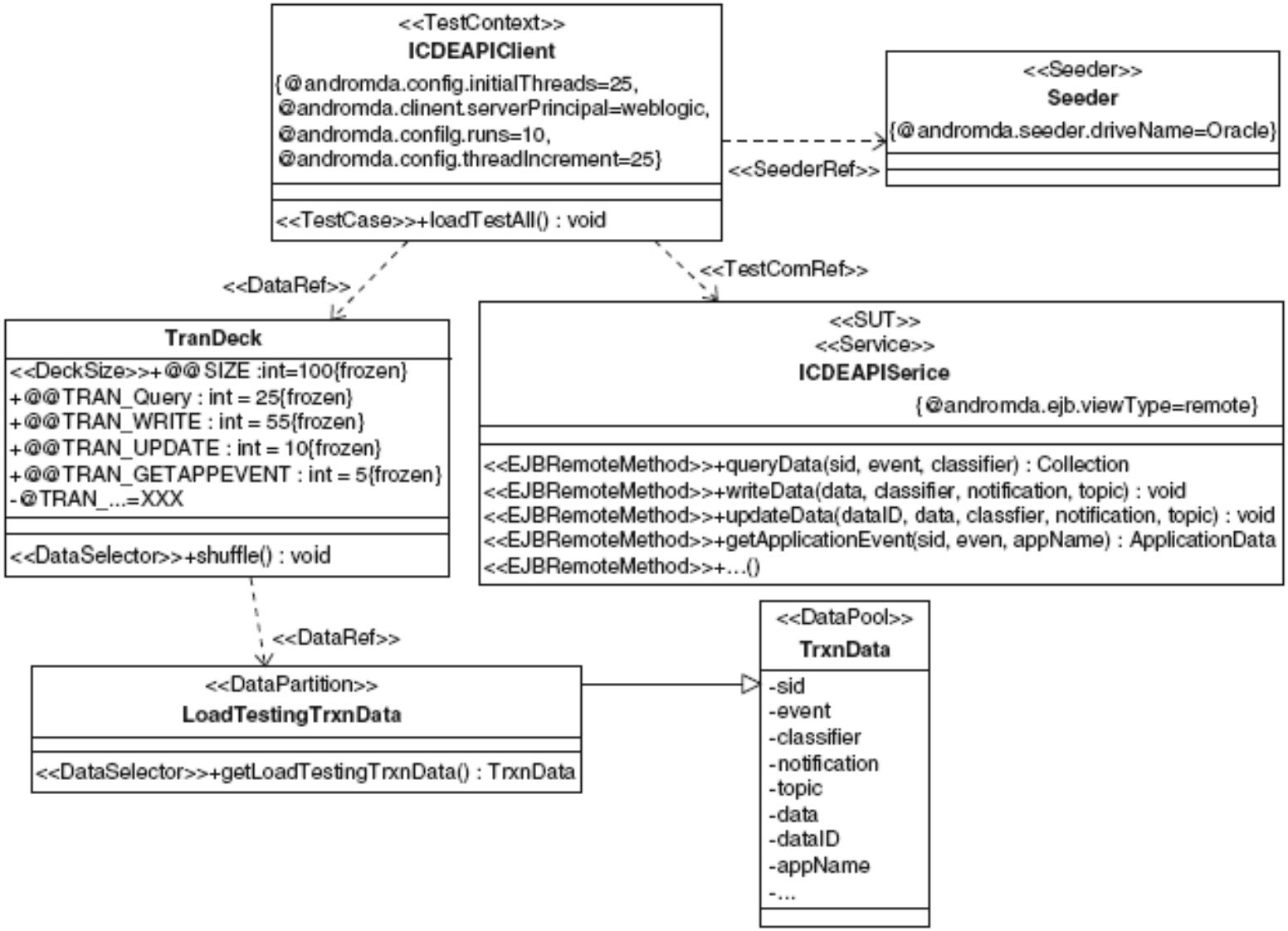
- MDA provides a generic application model and model mapping mechanism. MDA possesses portability, interoperability and reusability
- The reuse of code generation cartridges which are maintained by a large active user community with high quality is attractive
- Use MDA code generation cartridge could achieve site-specific features

# ICDE MDA-based Test Generator

- A UML profile and a tool are designed to automatically ICDE test suites from that specific description.
- The tool is built on top of an open source framework AndroMDA
- Input: UML based diagrams. Output: benchmark application including monitoring, profiling, and reporting utilities.


Application Design (UML) (Optional)
Tool Aided
Manual Abstraction
Benchmark app core Design (UML)
XMI output
UML 2.0 testing profile tailored for performance testing
Existing profiles (EJB, WS and etc)
AndroMDA Framework
Model Transformation
A new cartridge for the UML 2.0 testing profile
Existing cartridges (EJB, WS and etc)
Potential Integration
Monitoring Config files/scripts
Load Testing Suite
Core Benchmark
Performance Analysis Model
JMX monitoring
Validating or Providing Parameters
Prediction
Measurement
Our Extensions
Machine Generated
Existing Facilities

# Test Model

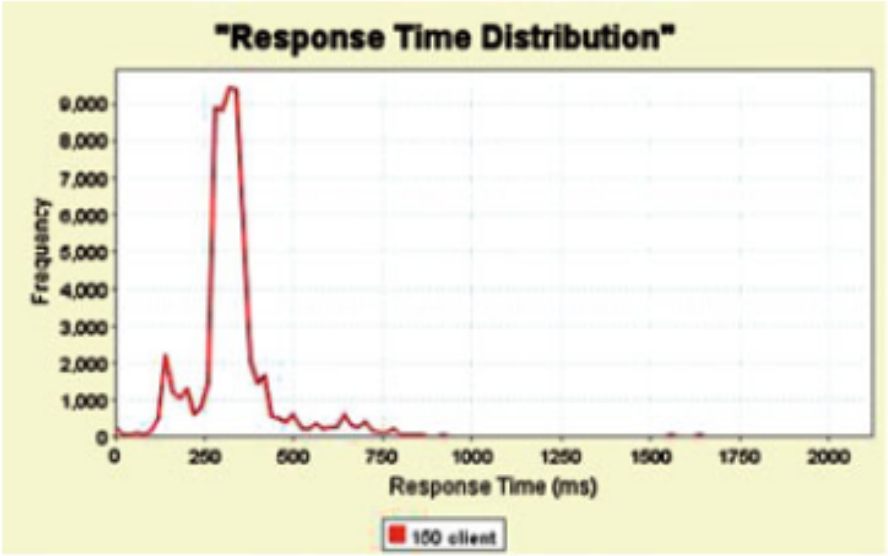

<<TestContext>>
ICDEAPIClient
{@andromda.config.initialThreads=25,
@andromda.clinent.serverPrincipal=weblogic,
@andromda.confilg.runs=10,
@andromda.config.threadIncrement=25}
<<TestCase>>+loadTestAll() : void
<<Seeder>>
Seeder
{@andromda.seeder.driveName=Oracle}
<<SeederRef>>
<<DataRef>>
<<TestComRef>>
TranDeck
<<DeckSize>>+@@SIZE :int=100{frozen}
+@@TRAN_Query : int = 25{frozen}
+@@TRAN_WRITE : int = 55{frozen}
+@@TRAN_UPDATE : int = 10{frozen}
+@@TRAN_GETAPPEVENT : int = 5{frozen}
-@TRAN_...=XXX
<<DataSelector>>+shuffle() : void
<<SUT>>
<<Service>>
ICDEAPISerice
{@andromda.ejb.viewType=remote}
<<EJBRemoteMethod>>+queryData(sid, event, classifier) : Collection
<<EJBRemoteMethod>>+writeData(data, classifier, notification, topic) : void
<<EJBRemoteMethod>>+updateData(dataID, data, classfier, notification, topic) : void
<<EJBRemoteMethod>>+getApplicationEvent(sid, even, appName) : ApplicationData
<<EJBRemoteMethod>>+...()
<<DataRef>>
<<DataPartition>>
LoadTestingTrxnData
<<DataSelector>>+getLoadTestingTrxnData() : TrxnData
<<DataPool>>
TrxnData
-sid
-event
-classifier
-notification
-topic
-data
-dataID
-appName
-...

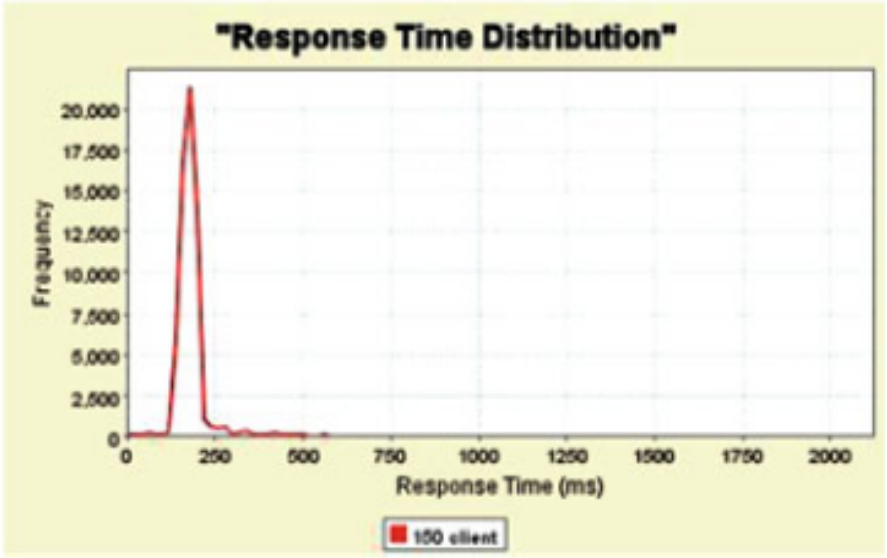
# Test Model Cont…

- ICDEAPIService : load entry point for test model
- ICDEAPIClient: consisting of a number of test cases
- TrxnData: test data used for calling ICDE APIs randomly generated to simulates the real work load of  the ICDE installation
- TranDEck: configure transaction mix for a test

# Test results

- Following figure shows the response time distribution for two different application servers. The workload is 150 concurrent users.



# Practical merits

- A large amount of time is save through automatically repetitive and error prone code generation for different platforms
- MDA raises the abstraction levels that make it easy to extend and be represented
- Seamless integration with other architectures not far away: e.g. high level semantic system integration and systems models transformation into low level SOA facilities

# References


- Essential Software Architecture 2nd, Ian Gorton, 2011 chapter 1-4 and chapter 14
- Software Architecture in Practice 2nd Len Bass Paul Clements, Rick Kazman, 2003
- L. Zhu, J. Liu, I. Gorton, N. B. Bui. Customized Benchmark Generation Using MDA. in Proceedings of the 5th Working IEEE / IFIP Conference on Software Architecture, Pittsburgh, November 2005
- Object Management Group: http://www.omg.org
- seminar on model-based software architecture.ppt
- An introduction to Model Driven Architecture. http://www.ibm.com/developerworks/rational/library/3100.html

# For further reading

- OMG, MDA Guide Version 1.0.1
  http://www.omg.org/mda/

- Thomas Stahl, Markus Voelter, Model-Driven Software Development: Technology, Engineering, Management, Wiley 2006

# Summary

- In this lecture, definitions of software architecture are first introduced in three different perspectives. Then modeling procedures and nonfunctional requirements compared with traditional functional designs  are given out.
- A specific software architecture: Model driven architecture, is analyzed in terms of its model transformation nature, unifying modeling language and three great features (portability, Interoperability, Reusability). Also a brief comparison between SOA and MDA shows a higher level abstraction feature of MDA.
- Finally, capacity planning and test on ICDE system is shown as a case study to take MDA into practice for meeting different platform requirements and environment constraints.

# Have a Nice Spring Break