J2EEML: Applying Model Driven Development to Autonomic Enterprise Java Bean Systems

Jules White

jules@dre.vanderbilt.edu

Institute for Software Integrated Systems (ISIS)
Vanderbilt University
Nashville, TN

Overview of Autonomic Computing

- *Autonomic computer systems aim to reduce the configuration, operational, and maintenance costs of distributed enterprise applications.*
- Autonomic systems manage themselves by:
  - **Monitoring** important state information
  - **Analyzing** the monitored state information to find potential indicators that the system needs to adapt
  - **Planning** how to adapt to recover from/advert unwanted states
  - **Executing** the adaptation plans to avoid entering or to leave the unwanted state
- Autonomic functionality reduces the amount of costly human intervention required to maintain the system

Challenges of Autonomic Applications

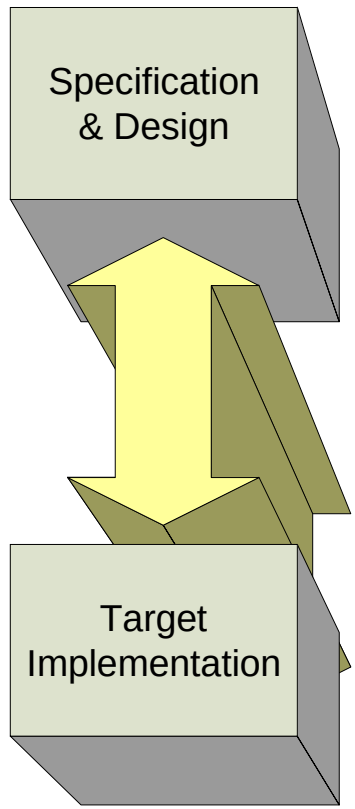
- Developing enterprise applications is already difficult
- Designing a flexible and efficient autonomic computing framework is difficult
  - Separating autonomic concerns from application logic
  - Monitoring and moving state information in an efficient and unobtrusive manner
  - Providing a flexibility points for adapting the application
- Hard to convince decision makers of the necessity of self-managing applications

J3 Process

- To make developing autonomic applications easier, we have developed the J3 Process which consists of:
  - J2EEML, a domain specific modeling language for describing autonomic EJB systems, their goals, and their adaptation plans
  - JFense, an autonomic computing framework for Java that significantly reduces the amount of code required for developing autonomic EJB applications
  - Jadapt, a J2EEML model interpreter that generates EJB implementation artifacts and JFense glue code to reduce development time and make developing autonomic systems more feasible

Motivation for Applying MDD to J2EE: Limitations with Implementation-centric Development

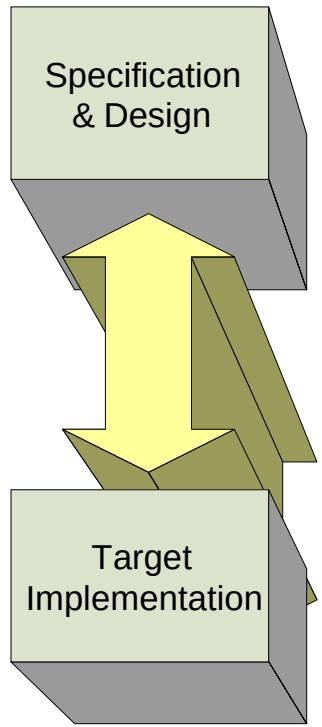
- Hard to synchronize the system design to the specification
- No assurance that the system design & implementation meet the QoS requirements
- No formal mapping from QoS properties to system components
- Developing each EJB requires maintaining multiple source files, interfaces, and XML descriptors, e.g. ejb-jar.xml, remote interface
- Hard to see the big picture, i.e. relationships between beans and relationships between beans and QoS requirements
- Design changes require significant code re-factoring
- Human written XML & programming language code is often buggy, e.g. JNDI naming errors



Connected through error-prone human interpretation

Benefits of Applying MDD to J2EE

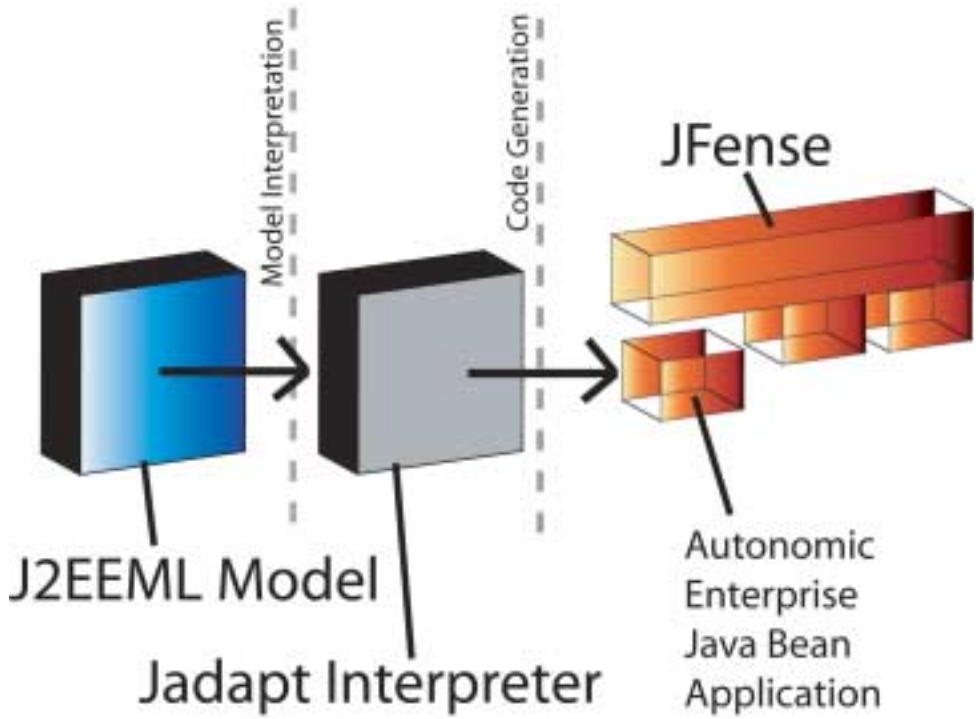
- Specification & design is done in notations specific to J2EE, easier to see the big picture
- Formally capturing the mapping from system design to QoS requirements, allows for:
  - Model analysis to ensure solution correctness
  - Clearer understanding of the relationships between system design & QoS requirements
- Implementations can be regenerated when design changes
- Autonomic QoS monitoring, analysis, planning, and adaptation code can be generated by the tool, which reduces development time
- The tool can synchronize the multiple artifacts required for each EJB to ensure that they are consistent
- The tool can generate test, build, and deployment infrastructure



Connected through automated translation

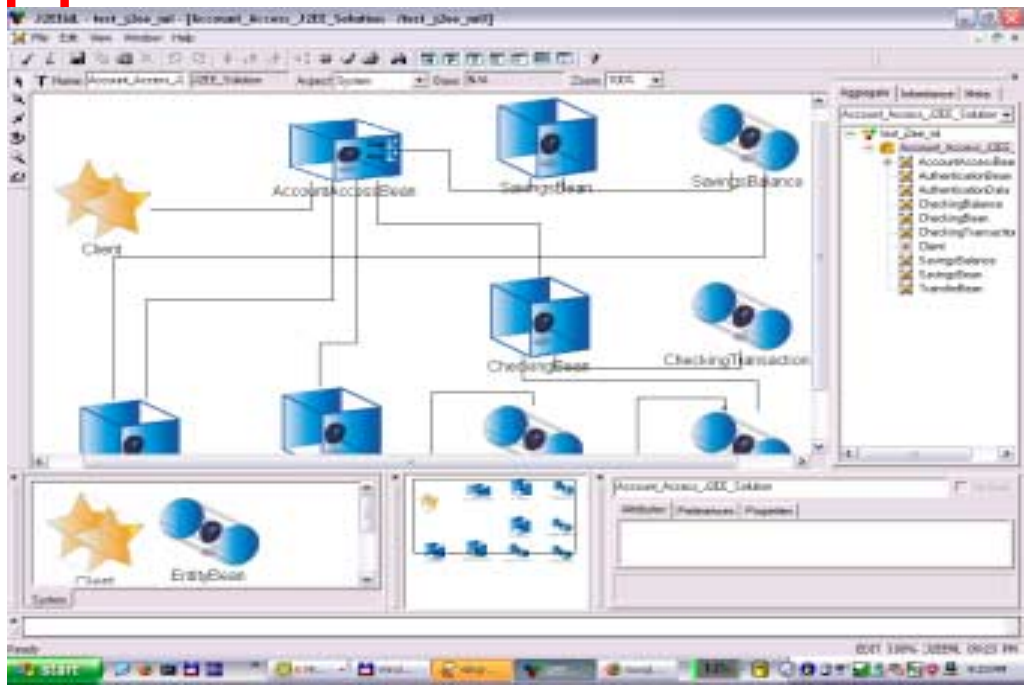
Overview of J2EEML, Jadapt, JFense

- System model is specified in J2EEML
- Jadapt model interpreter is run to generate the implementation artifacts (EJBs, deployment XML, etc.) and glue code to integrate with JFense
- The generated EJB application is run within the JFense framework to monitor, analyze, and adapt the application



Developing J2EE Applications with J2EEML

- We used GME to develop J2EEML, which is a DSML for Enterprise Java Beans (EJB) & their QoS requirements
  - Designers model the EJBs in the system
  - Designers create QoS goals that the system must achieve
  - Designers associate QoS goals with EJBs
  - Code generators produce the bean deployment descriptors, remote and local interfaces, utility classes, QoS monitoring & recovery framework, & build scripts for the application

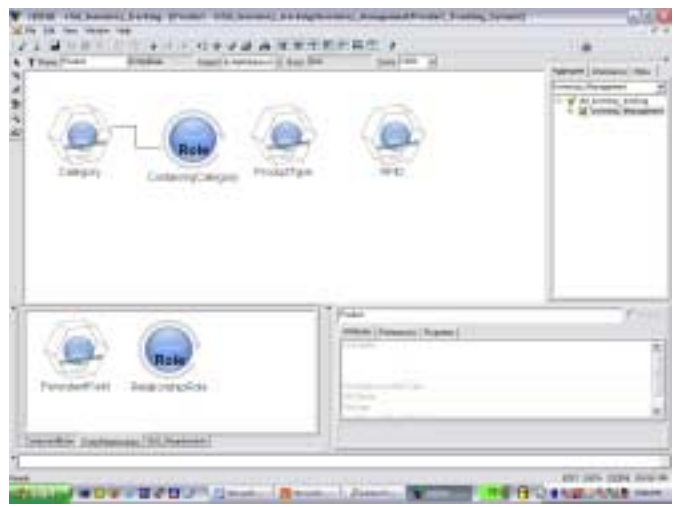


Key Benefits:

1. Guaranteed model correctness, e.g. JNDI naming
2. Generate bean implementations, monitoring code, & utilities
3. Generate build, deployment, & config infrastructure
4. Visualize complex bean interactions
5. Clear specification of the mapping from QoS reqs to system components

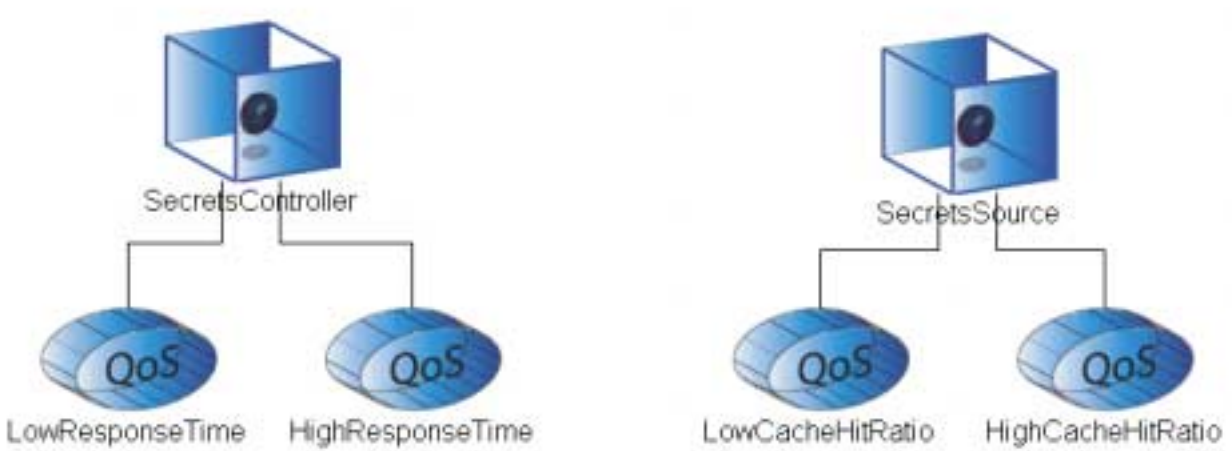
J2EEML Modeling Capabilities

- J2EEML provides EJB specific notation for:
  - *Session* and *Entity* beans
  - JNDI Naming
  - Remote and local interface composition
  - Method level security and transaction specification
  - Entity bean foreign key relationships
  - EJB creation and finder methods
  - EJBQL specification for *Entity* beans
  - Bean to Bean interactions
  - Method signatures
  - Documentation



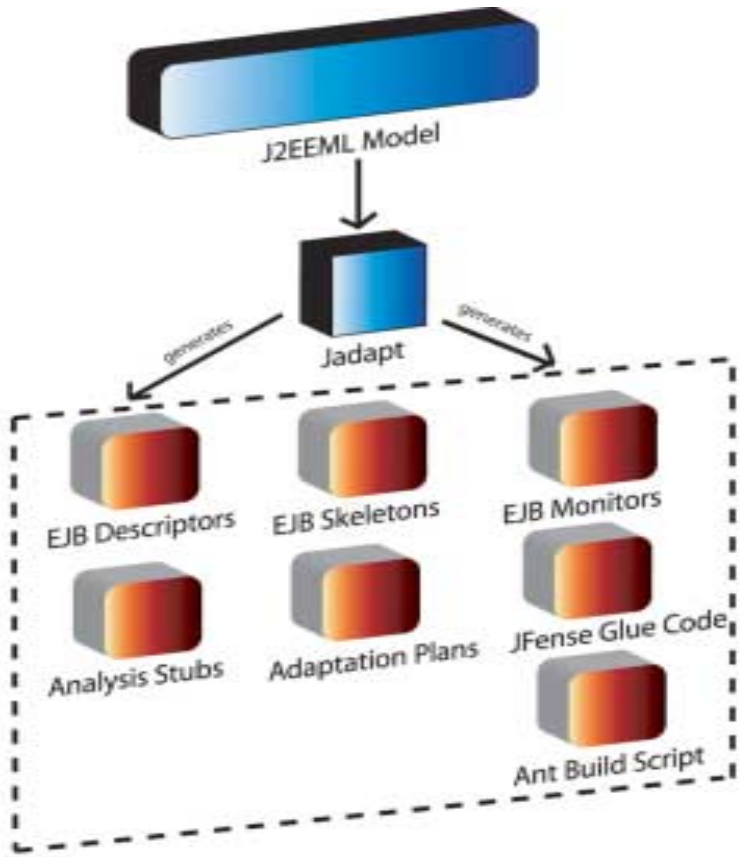
QoS / Autonomic Modeling Capabilities

- J2EEML provides abstractions for:
  - QoS goals
  - Adaptation plans to execute when goals are not being met
  - Adaptation actions that compose adaptation plans
  - Mappings between EJBs and QoS goals
  - Mappings between QoS goals and adaptation plans



Generating Autonomic J2EE Code

- J2EEML uses a model interpreter called Jadapt to generate:
  - Session and Entity beans
  - Remote and local homes and interfaces
  - Utility classes for JNDI lookups
  - Primary key classes and value object classes
  - QoS monitoring, analysis, and adaptation classes for use with JFense
  - EJB deployment descriptors
  - Another Neat Tool (ANT) build scripts
  - Client test suite skeletons
  - Javadoc documentation



Autonomic Design Decisions

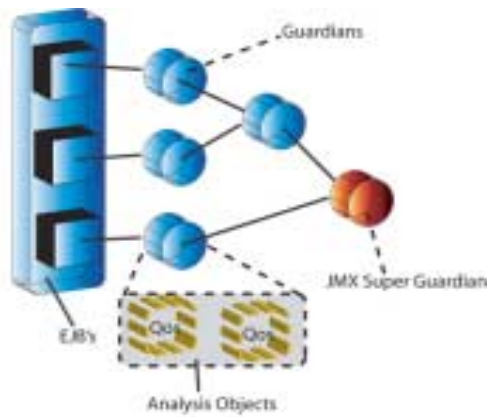
- Applications can choose between monolithic and layered autonomic systems.

- Monolithic autonomic systems use a single controller to monitor, analyze, plan, and issue adaptation orders
  - Can allow for more complex analysis of data from several sources
  - Can be complex and less flexible

- Layered autonomic systems use a hierarchy of controllers to provide autonomic functionality, each layer provides a higher and higher level of abstraction
  - More flexible, place recovery logic closer to the point of failure

Monitoring EJBs: The JFense Autonomic Framework

- JFense is an autonomic framework which provides standardized mechanisms for monitoring the QoS properties of EJBs and reacting to QoS failures
- Monitoring, analysis, planning, and adaptation code generated from J2EEML models can be plugged into JFense
- JFense has the ability to build multi-layered autonomic systems where monitoring, analysis, planning, and adaptation logic is distributed throughout the system
  - This gives developers the ability to place key autonomic logic closer to the managed component and react more quickly to failures
  - Complex QoS and autonomic properties can be built by decomposing the autonomic functionality into multiple layers of abstraction



Lifecycle Cost Analysis

- To motivate the need for autonomic applications and applying model driven development to them, we are performing a series of cost analyses on a sample system

- Evaluating performance cost of using autonomic systems
- Evaluating extra development cost incurred for autonomic systems
  - Measuring extra code (COCOMO models) and design complexity
    Evaluating recovery and downtime cost for autonomic vs. human intervention
- Evaluation of cost savings from using a DSML specific to J2EE as opposed to traditional development techniques

Autonomic Case Study

- Comparing different recovery mechanisms
  - We model a calculation that requires another component to complete its job
  - We compare recovery and downtimes for:
    - Fail-over style recovery, the calculation switches to use another equivalent component in the event that its current component becomes unavailable
    - Reset recover, the calculation attempts to reset the component in the event it becomes unavailable
    - Human recovery, the component is reset or restarted to fix the unavailable

Future Work

- Integration of J2EEML with the Eclipse's Draw2D framework
- Providing standardized mechanisms to allow the JFense framework to interoperate with Corba Component Model (CCM) applications
- Creation of complex autonomic capabilities between autonomic EJB systems through JMX
- Improvements to the generated autonomic framework to support Real Time Java (RTJ) and Distributed Real Time Java (DRTJ)
- Extension of J2EEML notation to include the web container and asynchronous messaging components of J2EE
- Complex model QoS analysis
- Refinements to existing constraints and modeling paradigm
- Roundtrip from model to code and code to model

J2EEML Project Information

- SourceForge site: http://www.sf.net/projects/j2eeml
- Papers: http://www.cs.wustl.edu/~schmidt/PDF/J2EEML.pdf