

Identifying Data Transfer Objects in EJB Applications

**Aleksandar Pantaleev
Atanas Rountev
Ohio State University**

Outline

- Background
 - DTOs in EJB applications
 - Motivation and challenges
- Dynamic analysis for DTO identification
 - Track accesses to object state in different EJB tiers
 - Implementation with JVMTI
- Experimental study
 - Real-world commercially deployed EJB application
 - Analysis cost and precision
- Conclusions and future work

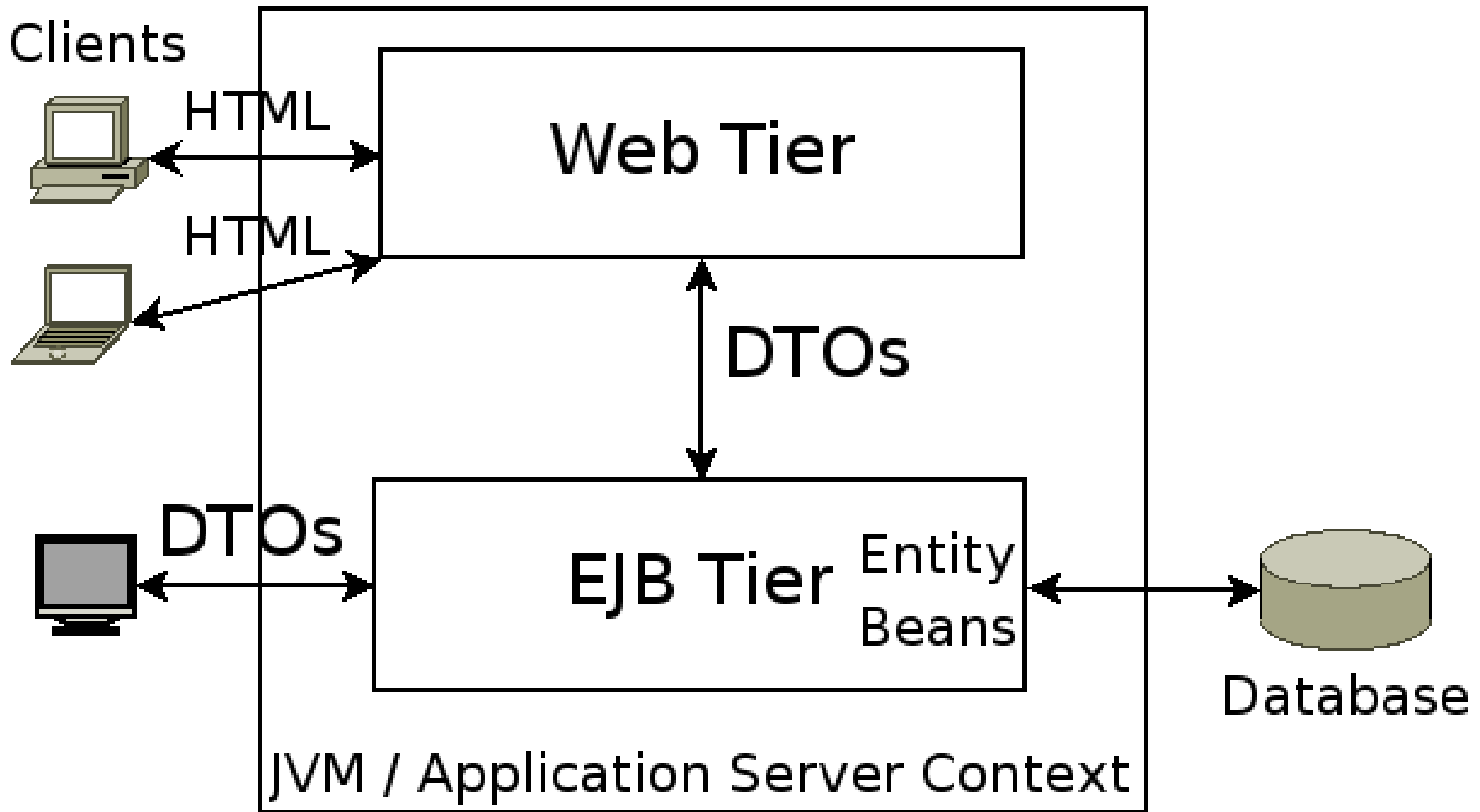
Data Transfer Objects

- DTO design pattern
 - Reduces the number of remote accesses
 - Often used in Enterprise Java software
- Goal: identify DTOs in an EJB application
 - Find classes whose instances implement the pattern
 - Approach based on dynamic analysis
- Motivation
 - Program comprehension and documentation
 - Performance optimizations
 - Software evolution (migrate from EJB2 to EJB3)

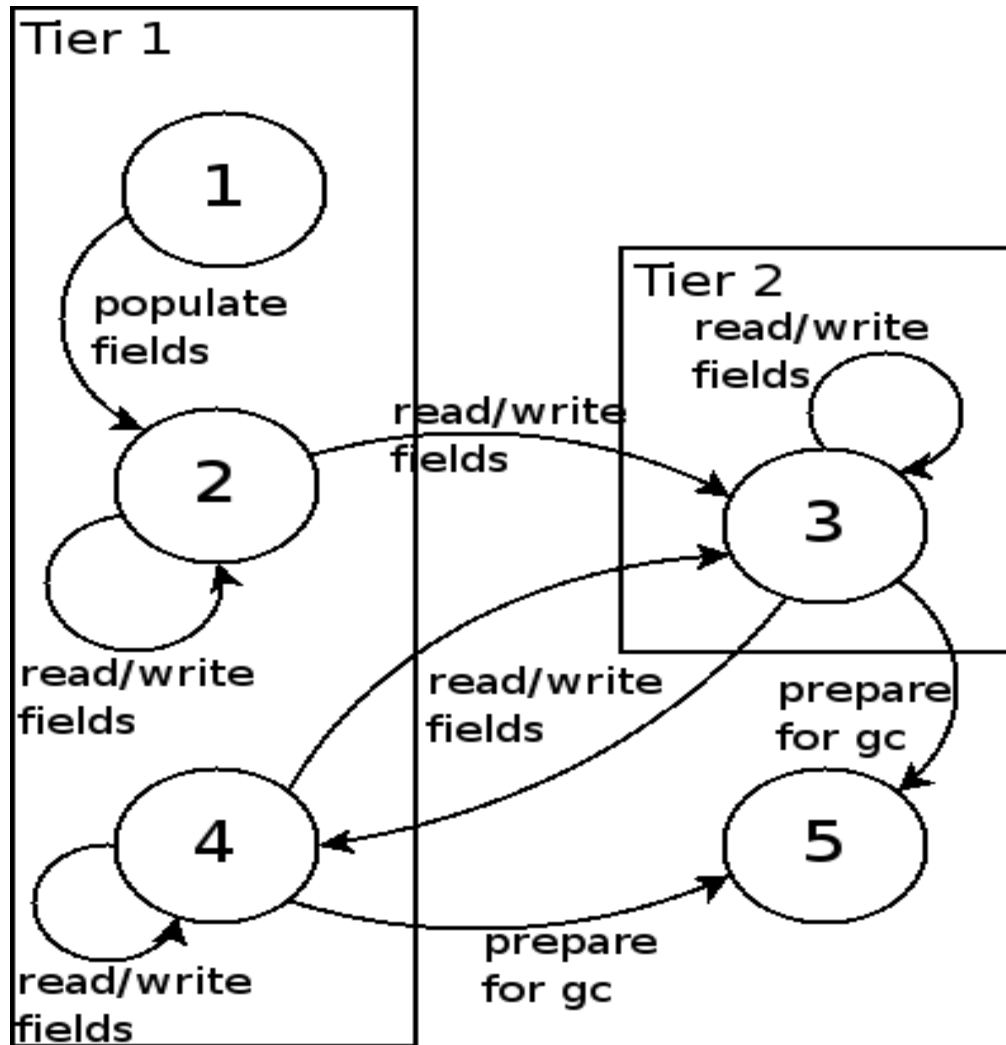
Sample DTO from the EJBCA Application

```
public class UserDataVO implements Serializable {
    private String username;
    private String subjectDN;
    private int caid;
    private String subjectAltName;
    private String subjectEmail;
    private String password;
    private int status;
    private int type;
    private int endentityprofileid;
    private int certificateprofileid;
    private Date timecreated;
    private Date timemodified;
    private int tokentype;
    private int hardtokenissuerid;
    private ExtendedInformation extendedinformation;
    public void setUsername(String user) { ... }
    public String getUsername() { ... }
    public void setDN(String dn) { ... }
    public String getDN() { ... }
    ...
}
```

Structure of an Enterprise Java Application



DTO Lifecycle



Dynamic Analysis

- Focuses on the state of objects
 - Tracks field reads & writes as opposed to method entries & exits
 - Fields of interesting (Serializable) objects tagged when the objects are created
- Matches state transitions against the DTO lifecycle
 - Requires precise identification of the EJB tier of the initiator of a state transition:
 - Handled by traversing the call stack
- Interested only in application classes
 - Robust with respect to reflection

Implementation

- JVMTI used with Java 6 JVM (agent written in C)
- JVMTI object tags used as pointers to analysis data structures
 - An object tag in JVMTI is a 32-bit integer
 - Supposed to be a user-assigned identifier
 - Agent written in C => we use that integer as a memory address
 - Cast between an integer and an address as needed
- Only interesting fields flagged as event triggers
- Garbage collection events sent only for tagged objects

Experimental Study

- EJB Certificate Authority (EJBCA) application
 - Open-source, commercially deployed
 - Comes with its own test suite
 - Multiple tiers: EJB, Web, console client, GUI desktop client
 - A total of 635 classes
- Number of classes loaded during testing
 - EJBCA's test suite was run with our JVMTI agent deployed
 - EJBCA version 3.4.1: 433
 - Application server (JBoss version 4.0.5): 1983
 - JDK version 1.6.0: 7137

Experimental Results

- 132 interesting (Serializable) classes in EJBCA code
- 13 deemed DTOs after manual inspection
- 11 of those 13 were utilized by the application's test suite (loaded in the JVM)
- Analysis precision: big picture
 - One false positive
 - One false negative

False Positive and False Negative

- False positive: not really false
 - Carries state
 - Problem is, never allows direct access to its state, so technically it is not a DTO
- False negative: wrapped by another DTO
 - True DTO
 - All reads & writes of its fields carried out through the wrapper
 - Wrapper classified as DTO, but wrapped object appears to have never moved between tiers

Run-time Performance

- Start-up overhead
 - 1m32s without agent
 - 91% overhead with full agent deployed
 - 29% overhead with dummy agent (JVMTI capabilities enabled, but all agent event handlers return immediately)
- Application overhead
 - 4m53s without agent
 - 263% overhead with full agent deployed
 - 31% with dummy agent
- Also tested with method entry & exit events
 - Start-up took ~2.5h
 - Application left overnight, had not finished

Future Work

- Improve the algorithm
 - performance
 - precision
- Migrate code: EJB2 to EJB3
 - EJB2 Entity beans are tightly coupled with DB
 - State is moved exclusively with DTOs
 - EJB3 Entity beans can be detached from the DB
 - Merge EJB2 Entity beans and their corresponding DTOs to form EJB3 Entity beans
- Identify other interesting design patterns in Enterprise Java applications
 - e.g., Data Access Object (DAO)

Questions?