State Is Not Evil

Dan Johnsson
Omegapoint AB / Frobozz AB; Sweden

dan.johnsson@omegapoint.se / dan.johnsson@frobozz.se
Some Opinions on State

- State is not evil, it is just misunderstood
  - Carl Jung
- State is not evil, it is the love of state that is evil
  - Book of Job
- The peoples’ right to bear state must not be infringed upon
  - Constitution of the United States
- If state is outlawed, only outlaws will carry state
  - Librarian party
- There is something rotten in the state …
  - Shaxbeard
- “I’ll be back!”
  - Governor of the state …
Silly Example - Bar

Behind Counter

Bar

Ladies

Gents
Bar Activities

- Move around
  - Bar area
  - Behind counter
  - Ladies
  - Gents
  - Home

- Buy Drinks
  - only in bar area

- Take money from cash register
  - only behind counter

Where am I? What can I do?
BarDweller - EJB Session

- Interface to clients

```java
public interface BarDweller extends EJBObject {
    public void buyDrink() throws RemoteException;
    public void takeMoney() throws RemoteException;
    public void goBar() throws RemoteException;
    public void goBehindCounter() throws RemoteException;
    public void goLadies() throws RemoteException;
    public void goGents() throws RemoteException;
    public void goHome() throws RemoteException;
}
```

- Implementation will enforce state rules
Realistic Example

- Dance school
  - Single (Tap, Hip Hop) and pair dances (Swing)
  - Single or couple registration
  - Quota for men/women

- Strategies for registration ("complex order")
  - Form strategy
    - Gather all info and try, and try again, and again ...
  - Personal service strategy
    - Search with fast fail and partial redo
This Presentation

- **Scope**
  - Management of client state in enterprise systems
  - Session Enterprise JavaBeans

- **Depth**
  - Recognition
  - Rephrase
  - Apply
  - Analysis
Outline

- Philosophy - state *etc.*
- Technology - Session EJBs
- Myth - SFSB *vs.* SLSB
- Closer analysis
- Alternative architectures
Philosophy

- Enterprise Systems
- State - from where?
- The Cube
- Kinds of State
Enterprise Systems

- Information system
  - Information = data *about* something
  - Manage data
    - Oppose: games, FEM-calculations
  - Producers and consumers
  - Consistency of data
  - Enterprise Logic = rules on data
Large Scale Distributed Enterprise Systems

- **Large Scale**
  - Lots of users
  - Lots of kinds of users
  - Lots of functionality

- **Distributed**
  - No control of clients
  - Security challenges
All Computer Systems Are Simulations

- Simulation of what is
  - Aerodynamics of airplane wing
- Simulations of what should have been
  - File cabinet w/ customer info in office
- Naive Model
  - Bank account = jar with money
States

- OOA: data and usage objects
- Data - persistent data in system
- Usage - client sessions
- Both have state
- Our focus: session objects
State - from Where?

- Earlier: human interaction
  - Ex: dance class registration
  - Humans remember things
  - "State of conversation"

- Now: system interaction
  - Want same user experience
  - Sessions with "conversational state"

- State as we are humans
Conversational State

- Experienced by client
- Property of application

fu?

bar!
The Cube - Tiers

- Split system functionality
  - Client
  - Presentation
  - Enterprise/Business Logic
  - Integration
  - Resource

```
 CLIENT    PRESENT    LOGIC    INTEGR    RES
 C          P          L          I          E
 L          E          O          G          R
 I          G          R          E          S
 00110011
 00101011
 11000110
```

OOA
### The Cube - Layers

- Layers of abstraction

<table>
<thead>
<tr>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual Platform</td>
</tr>
<tr>
<td>Operating Environment</td>
</tr>
<tr>
<td>OS</td>
</tr>
</tbody>
</table>
### The Cube - 2D

<table>
<thead>
<tr>
<th>check-date.js</th>
<th>seats.jsp</th>
<th>Ticket</th>
<th>SELECT ...</th>
<th>UPDATE ...</th>
<th>bookings</th>
</tr>
</thead>
<tbody>
<tr>
<td>JavaScript</td>
<td>Servlet 2.3</td>
<td>EJB 2.0</td>
<td>JDBC 2.0</td>
<td>SQL 92</td>
<td></td>
</tr>
<tr>
<td>JSP 1.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mozilla</td>
<td>Tomcat 4</td>
<td>JBoss</td>
<td>Connector/J</td>
<td>MySQL</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenBSD</td>
<td>BeOS</td>
<td>Linux</td>
<td>Linux</td>
<td>Linux</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Cube - Capabilities

- Emergent properties
  - Performance
  - Availability
  - Security
  - Extensibility

- Not located in single component
- Discussing architecture - must consider entire system
Some Client State Is Presentation State

- Presentation logic is for info-gathering flow
- *e.g.* On-line registration
  - Alt 1: One massive registration
    - all info gathered in the same page
  - Alt 2: Wizard
    - same info gathered in steps (name, address, ...)
    - can even have branches (US or Swedish address)
- No business logic difference!
  - one business logic call: “register(RegInfo)”
Some Client State Is Business State

- Business logic is for guaranteed consistency
- Stateful conversation can have branches
  - conversation can take different routes
  - consistency w model still guaranteed
    - e.g. cannot continue registration if course is full
- Course grained methods
  - course grained state changes
Are Not All Client States Presentation?

- PL for info-gathering flow
- BL for enforcing enterprise rules
  - guarantees consistency
- Distinction - BL if
  - Transaction support
  - Security constraint
  - Dependent on system state
PL vs. BL  ex: Dance School

- **Presentation**
  - Wizard
  - Name
  - Address
  - Phone

- **Business**
  - Choose activity
  - Register partner
  - Choose lead/follow
Recap

- Must consider entire system
- State property of system
- State as we are humans
- There are true business states
Outline

- Philosophy - state *etc.*
- Technology - Session EJBs
- Myth - SFSB *vs.* SLSB
- Closer analysis
- Alternative architectures
Session EJBs

- Enforce cross entity consistency
  - Ex: transaction between accounts

- Enforce business flow
  - Ex: payment info before shipping

- "Top level" services to user tiers

- Other uses
  - Infrastructure services: mail, log etc.
  - Not focus
Stateful - Stateless

- Confusion
- All sessions have state
  - Bank accounts have balance, even if zero
- Conversational state property of code
- SFSB, SLSB are deploy options
- SLSB will only work for ∅-state beans
Lifecycle of Session EJB

- Two deploy options
  - SFSB - for all session
  - SLSB - for Ø-state sessions
SFSB - Deploy Option

- Clients have private instances
- Full memory - swap to disk
- Penalty: swap time delay
- Instances = simultaneous sessions
SLSB - Deploy Option

- EJB container allowed party trick
- Instance not occupied between calls
- Instance is occupied during calls
  - Cf. servlets are truly multi-threaded
- Pool size = simultaneous calls
Recap

- Session EJBs represent top level services
- State is property of code
- SFSB and SLSB are deploy options
- SFSB have swap mechanism
- SLSB does not need swap
Outline

- Philosophy - state *etc.*
- Technology - Session EJBs
- **Myth - SFSB vs. SLSB**
- Closer analysis
- Alternative architectures
Myth

- “Thou shalt always use SLSB”
- Take up less resources
  - True: Less memory on EJB server
- Higher performance
  - True: No swapping overhead
Outline

- Philosophy - state *etc.*
- Technology - Session EJBs
- Myth - SFSB vs. SLSB
- Closer analysis
- Alternative architectures
Closer Analysis

- Myth simplifies
  - Misses system view (remember “entire system”)
  - When are SFSB troublesome?
  - How big is the problem?

- Nuisances
  - Memory consumption
  - Swap-time delay
How Big Is the Problem?

- **Intensity of penalty**
  - Not every time - only on misses
  - 125% memory usage - miss in 20% of calls
  - 200% memory usage - miss in 50% of calls

- **Size of penalty**
  - one passivate + one activate + some admin
  - *i.e.* two disc accesses w/ serialisation < 100 ms
Some Bar-Scenarios

- Different bar patrons
  - Fast party animals
  - Shanti bar dwellers
- Few animals running around - OK
- Many dwellers moving slowly - OK
- Extremely many dwellers - Crowded!
- Many dwellers and few animals - OK
- Many animals - Gahd! Get out of here!
Some Scenarios

- Different client characteristics
  - Few fast sessions
  - Many slow sessions
  - Extremely many slow sessions
  - Many slow and few fast
  - Many fast

- What does the client experience?
Few Fast

- *e.g.*: clerks within a company, other systems
- Can stay in memory
- No swapping
- No problem
Many Slow

- *e.g.:* human Web users
- Memory consumption
  - Swapped out
- Swap time delay
  - Marginal
  - Swap circle < 100 ms
  - Roundtrip time > 1 s
- No Problem
Extremely Many Slow

- *e.g.*: under dimensioned system
- Swap admin becomes heavy
- Less time for real work
- Downwards spiral
- **Problem!**
Many Slow and Few Fast

- *e.g.*: Other subsystems, B2B integration
- Memory consumption
  - Majority swapped out
- Swap time delay
  - Eager sessions never swapped out
  - (sound implementation)
- No Problem
Many Fast

- *e.g.*: lots of connected subsystems
- Very fast swaps needed
- Swap admin becomes heavy
- Similar to “Extremely Many Slow”
- Problem!
Summary of Scenarios

- Few Fast - OK
- Many Slow - OK
- Extremely Many Slow - Problem
- Many Slow and Few Fast - OK
- Many Fast - Problem
Solutions to Problem Scenarios

- Changing architecture
  - Let’s investigate options
  - Expensive
    - To do
    - To keep

- Changing operating environment
  - More memory
    - reduces intensity
  - Faster I/O
    - reduces penalty
Recap

- Myth takes too narrow view
  - Misses system view - does not consider clients
  - Not swap all the time
  - Swap is not extremely expensive
- SFSB no problem in several usual scenarios
Outline

- Philosophy - state *etc.*
- Technology - Session EJBs
- Myth - SFSB *vs.* SLSB
- Closer analysis
- **Alternative architectures**
Alternative Architectures

- Cannot remove state
  - State property of application
  - Like it or not - it’s still there

- Can push state elsewhere
  - Into database
  - Onto web server
  - Onto client
    - GUI client
    - Web client
Capabilities

- Performance
  - Response time
- Capacity
  - Simultaneous usage
- Scalability
  - To increase capacity
- Reliability
  - Consistency of behaviour
- Exensibility
  - Add functionality
- Security
  - Fulfil security policy

- Architecture is always a trade-off
  - Change architecture: Buy one, pay with another
Architectural Transformation

- Change architecture
  - Gain capability
  - Will effect other capabilities
- No way of knowing result (too complex)
  - Measure
  - Transform
  - Measure again
Some Transformations

- Optimisation
  - gain performance
  - lose extensibility

- Optimistic locking
  - gain performance
  - lose reliability

- Clustering
  - gain availability
  - lose manageability

- Encryption
  - gain security
  - lose performance and capacity

- Architecture is always a trade-off
Shopping Cart

- Criticised Example
  - Carts in reality are transient
  - Carts in e-stores are persistent
- Simple and standard

- This version
  - transient
  - keeps client name
  - items represented by int:s
  - ‘add’ checks inventory
  - fast fail
  - ‘buy’ makes persistent
State in Session EJB - Client

- **Client**
  - state
    ```java
    private Cart cart;
    ```
  - init
    ```java
    cart = carthome.create(name);
    ```
  - add-action
    ```java
    cart.add(item);
    int sum = cart.sum();
    ```
  - buy-action
    ```java
    cart.buy();
    cart.remove();
    ```

- **CartHome**
  - public Cart create(String name)...

- **Cart**
  - public void add(int item)...
  - public int sum()...
  - public void buy()...
State in Session EJB - Bean

- CartEJBean
  - private String name;
  - private List items;
  - public void ejbCreate(String name)
    - this.name = name;
    - this.items = new ArrayList();
  
- public void add(int item) {
    - items.add(new Integer(item));
  }

- public int sum() {
    - int result = 0;
    - // items iteration
    - return result;
  }

- public void buy() {
    - // commit items
    - this.items = null;
  }
State in Session EJB - Analysis

- Performance
  - OK, as long as swapping under control

- Capacity
  - Craves memory for many sessions, least swapping

- Extensibility
  - Easy to understand
  - Truthful to model

- Reliability
  - No problem

- Security
  - Easy to verify
State in Database

- Pass client ID to SLSB
- SLSB updates state in DB
- BL still in BL tier
- Client must handle ID
- Client must initialize
- Client must remove explicitly
  - Session EJB has timeout - can give orphan data
State in Database - Client

- **Client**
  - **state**
    ```java
dprivate Cart cart;
dprivate int id;
  
- **init**
  ```java
  cart = carthome.create();
id = cart.createCart(name);
```

- **add-action**
  ```java
  cart.add(id, item);
  int sum = cart.sum(id);
```

- **buy-action**
  ```java
  cart.buy(id);
cart.removeCart(id);
cart.remove();
```

- **CartHome**
  - public Cart create()...

- **Cart**
  - public int createCart(String name)...
  - public void removeCart(int id) ...
  - public void add(int id, int item)...
  - public int sum(int id)...
  - public void buy(int id)...

```java
```
State in Database - Bean

- **CartEJBean**

  ```java
  public void ejbCreate() {}
  public int createCart(String name) {
    // JDBC: INSERT INTO session
    return id;
  }
  public void removeCart(int id) {
    // DELETE FROM session
    // DELETE FROM sessionitem
  }
  public void add(int id, int item) {
    // INSERT INTO sessionitem
  }
  public int sum(int id) {
    // SELECT sum(item)
    // FROM sessionitem WHERE
    id=?
  }
  public void buy(int id) {
    // SELECT name FROM session
    ...
    // SELECT ... FROM sessionitem...
    // commit items
  }
  ```
State in Database - Analysis

- Performance (-)
  - swap
  - DB access
    - memory cache
- Capacity +
  - Lot less memory

- Extensibility -
  - Less natural interface
    - create does not start a session
    - createCart does

- Reliability

- Security
  - BL still protected
State in HttpSession

- Business logic in Web tier code?
  - Effect: no session logic in EJB tier
  - EJB tier no longer guarantees consistency
  - Problems with transactions (e.g. atomic ‘buy’)

- Just keep state in Web tier
  - Send state to EJB for state changes
  - EJB perform state change actions
  - Updated state as return value
State in HttpSession - Client

- **Client**
  - **state**
    - private Cart cart;
    - private CartState state;
  - **init**
    - cart = carthome.create();
    - state = cart.create(name);
  - **add-action**
    - state = cart.add(state, item);
    - int sum = cart.sum(state);
  - **buy-action**
    - state = cart.buy(state);

- **CartHome**
  - public Cart create()...

- **Cart**
  - public CartState
    - create(String name)...
    - BL: customer verification
  - public CartState
    - add(CartState state, int item)...
    - BL: check inventory
  - public int sum(CartState state)...
    - BL: discounts
  - public CartState buy(CartState state)....
State in HttpSession - Bean

- CartEJBean
  - public void ejbCreate() {}
  - public CartState create(String name) {
    return new CartState(name);
  }
  - public CartState add(CartState state, int item) {
    return state.add(item)
  }
  - public int sum(CartState state) {
    int result = 0;
    for(Iterator iter = state.items(); ... ) {
      Integer item = ...
      result += item.intValue();
    }
    return result;
  }
  - public CartState buy(CartState state) {
    for(Iterator iter = state.items(); ... ) {
      Integer item = ...
      // commit items
    }
    return state.empty();
  }
State in HttpSession - State

- **CartState - data holder**
  - private String name;
  - private List items = new ArrayList();
  - public CartState(String name) {
      this.name = name;
  }
  - public CartState add(int item) {
      items.add(new Integer(item));
      return this;
  }
  - public Iterator items() {
      return items.iterator();
  }
State in HttpSession - Analysis

- **Performance**
- **Capacity (+)**
  - Same amount of memory
  - HttpSession might be more lightweight
- **Extensibility** -
  - have to pass state
- **Reliability** -
  - no guarantee of data consistency
- **Security** -
  - State managed outside server
  - Malicious client could change state
State in GUI Application

- State completely pushed from server
- Code = HttpSession case
- Business logic
  - still server round-trip
  - (BL in GUI/HttpSession => EJB meaningless)
State in GUI - Analysis

- Performance -
  - State transferred over slow cable
- Capacity ++
  - No trace of state on Web- or EJB-server
- Extensibility -
- Reliability
- Security --
  - State exposed on client deployed “elsewhere”
State in Cookie

- State pushed completely off server in Web environment
- State has to be encoded as String
  - String encode(CartState)
  - CartState decode(String)
State in Cookie - Analysis

- Performance --
  - Really slow to ship full text-encoded state
- Capacity ++
- Extensibility -
- Reliability
- Security --
  - Cookies can very easily be manipulated
Summary - Alternatives

- State in database
  + capacity, (-) performance, - extensibility

- State in HttpSession
  (+) capacity, - extensibility, - security

- State in GUI application
  ++ capacity, - performance, -- security

- State in Cookie
  Forget that! Perhaps small, trivial state
How to Know? Test!

- Check out sample code
  - `pserver:cvs.frobozz.se:/repository/public`
  - `module: css2003state`
  - `username: css2003, password: kovsky`

- Test it on your app server (laptop?)
- Enhance (Web tier, profiling scripts, etc.)
- Share your work! Commit!
- Licence: completely unrestricted (both ways)
Conclusion

- No alternative obviously better
  - However: architecture is always a trade-off
- State in session EJB first alternative
- Investigate alternatives
  - Measure
  - Make architectural transformation
  - Measure again
- However: State is Not Evil!