Platform-independent UML Specification of Enterprise Applications in Project Ace

Bruce Daniels

Project Ace

Sun Microsystems Labs

Sun microsystems

We make the net work.

Talk Outline

- Problems with Enterprise Applications
- New Approach Needed
  - Application Specification
    - Persistent Data (Business Objects)
    - Data Usage (Business Process Tasks)
  - Implementation Generation
    - Architecture Choice & Performance Optimization
- Results

Problems with Creation of Enterprise Applications

- Programming is too complex = demands very high developer skills
- Time-to-market is too slow = longer than competitive business cycle
- Development is too rigid = incapable of early prototyping & unresponsive to technology changes

What is needed for Enterprise Applications?

- High-Level Specification of Application Intent
  - Architecture Independent
    => hides complexity & demands less skills of staff
  - Concise => accelerates time-to-market 10X
  - Flexible => supports prototyping & higher satisfaction
- Generation of Application Implementation
  - Migration => supports multiple architectures
  - Optimization => provides high performance

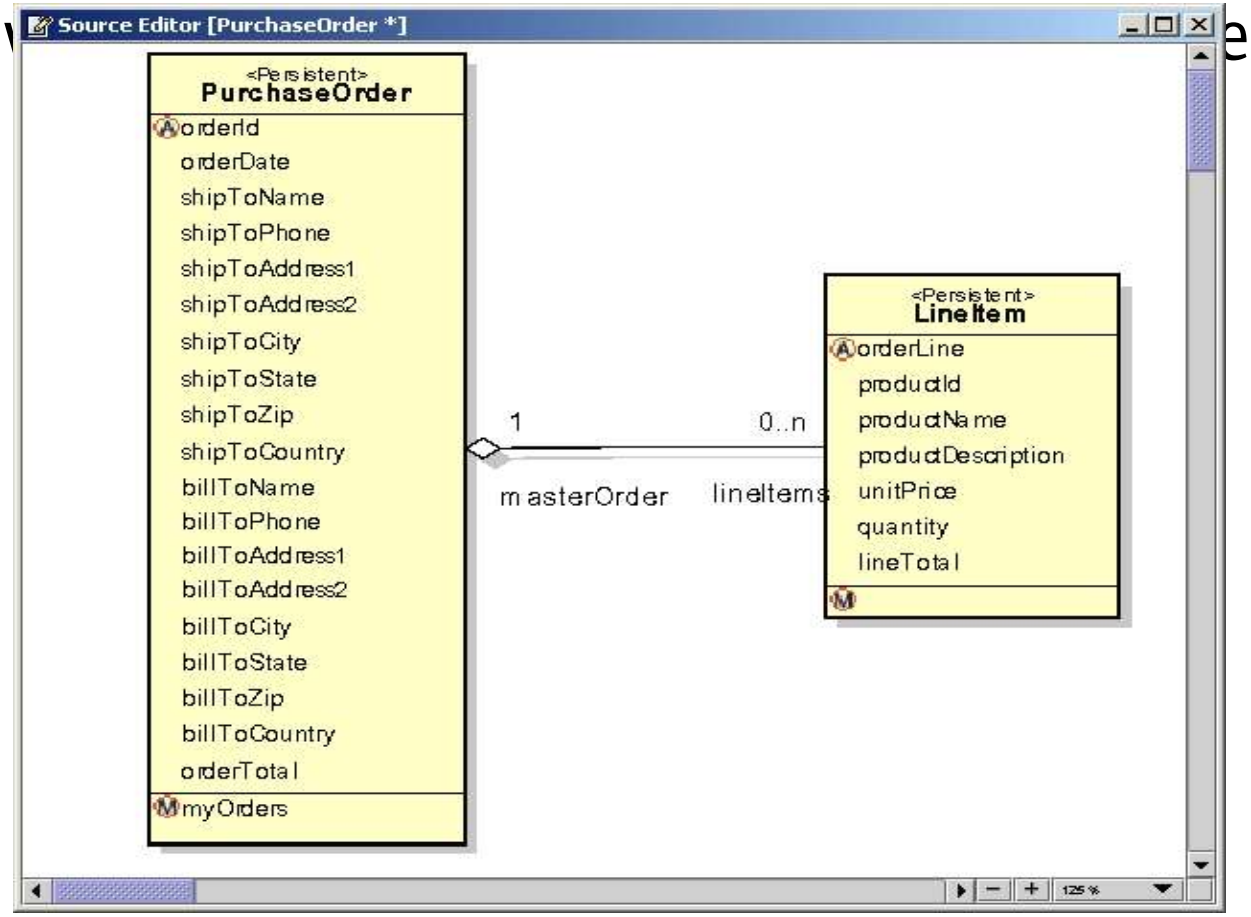
Specification ≠ Model

- Specification
  - detailed and exact definition of a particular system to be built
- Model
  - tentative representation of certain selected aspects of a system
  - often incomplete and inexact, e.g. a model car
- We Need an Application Specification!

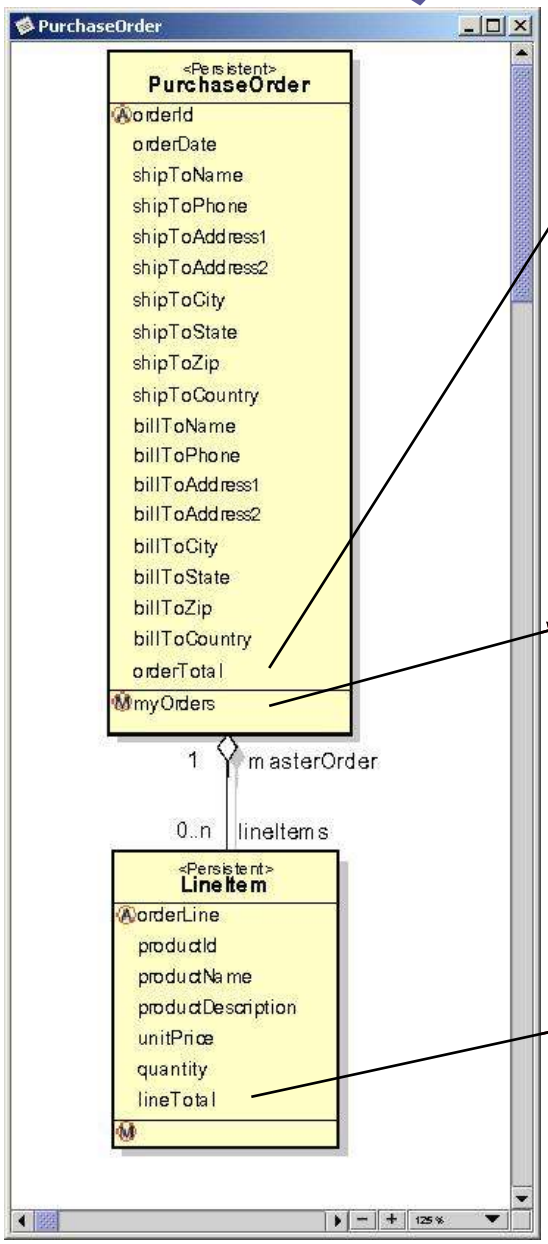
Specification 1: Persistent Data

- Business object & relationship UML diagram

w eries



Biz Object Logic Required



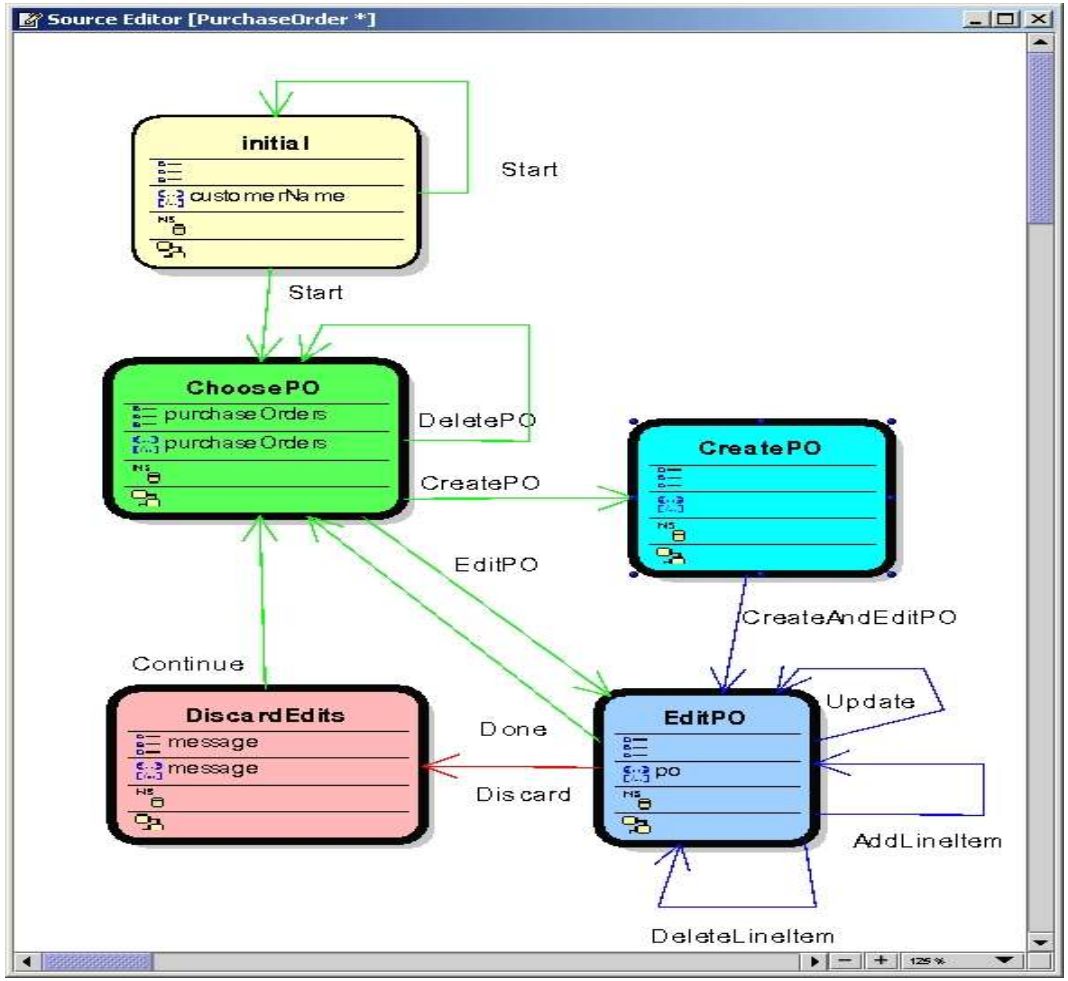

```
Computed Decimal orderTotal = {
    BigDecimal total = 0;
    for(LineItem li : lineItems)
        total += li.lineTotal;
    return total;
}
```

```
factory method myOrders(String custName)
          returns List of PurchaseOrder
    QUERY ( SELECT p
              FROM PurchaseOrder p
              WHERE billToName = :custName )
```

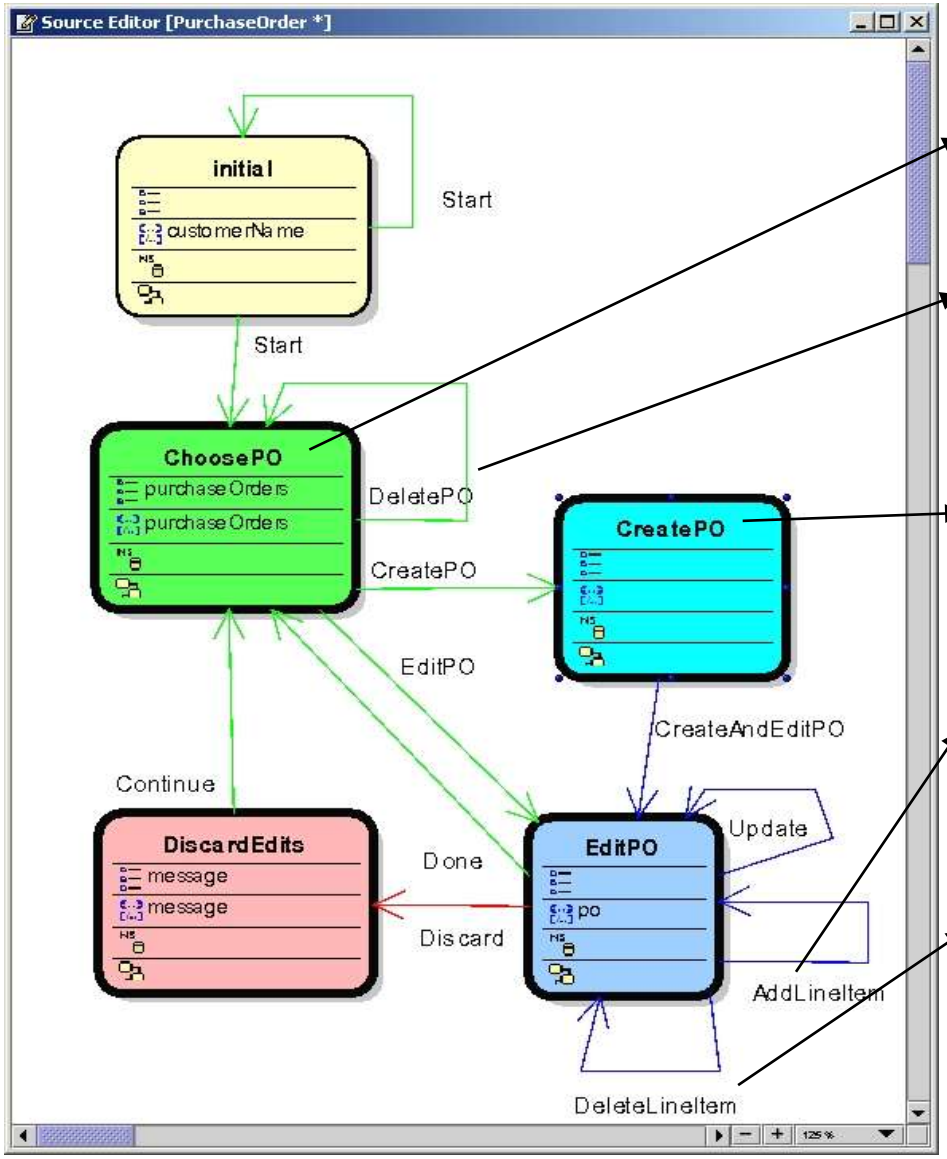
```
computed Decimal lineTotal =
        (unitPrice * quantity);
```

Specification 2: Data Usage

- Business Process task & transition UML State diagram with object access, user interactions, & business logic calls



Task Logic for Connection To Objects




```
PurchaseOrders =
        PurchaseOrder.myOrders(customerName);
```

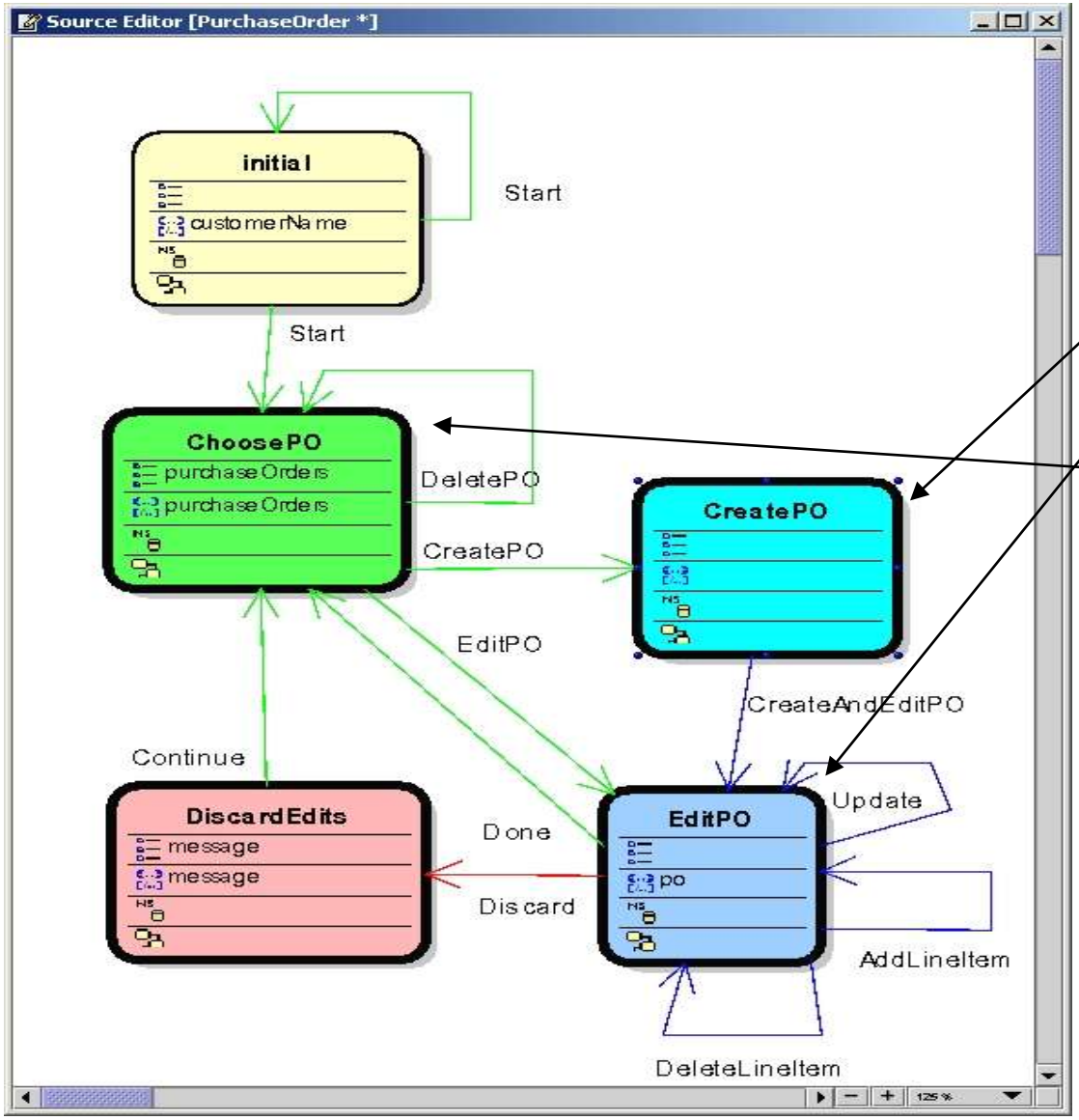
```
PurchaseOrder po = purchaseOrders.getSelectedOne();
PurchaseOrder.remove(po);
return new ChoosePO();
```

```
PurchaseOrderPK pk = PurchaseOrder.assignNextPK();
PurchaseOrder po = new persistent PurchaseOrder(pk);
return new EditPO(po);
```

```
LineItemPK pk = LineItem.assignNextPK(po);
LineItem li = new persistent LineItem(pk);
return new EditPO(po);
```

```
LineItem li = po.lineItems.getSelectedOne();
LineItem.remove(li);
return new EditPO(po);
```

Specify Task Transactions



transaction edit {
CreatePO,
EditPO
commits to
ChoosePO
}

Specifications can be textual -
with bi-directional editing with diagrams

```
• persistent class PurchaseOrder {
        persistent Long orderId;
        persistent Date orderDate = new Date()
        persistent String shipToName
        persistent String shipToPhone
        persistent String shipToAddress1
        persistent String shipToCity
        persistent String shipToState;
        persistent String shipToZip;
        persistent String shipToCountry = "USA"
        computed Decimal orderTotal =
              {
                 BigDecimal total = 0;
                 for(LineItem li : lineItems)
                    total += li.lineTotal;
                 return total;
              }
        owns LineItem lineItems(0,n)
                       inverse masterOrder (1,1);
        PRIMARY KEY (orderId);
}
```

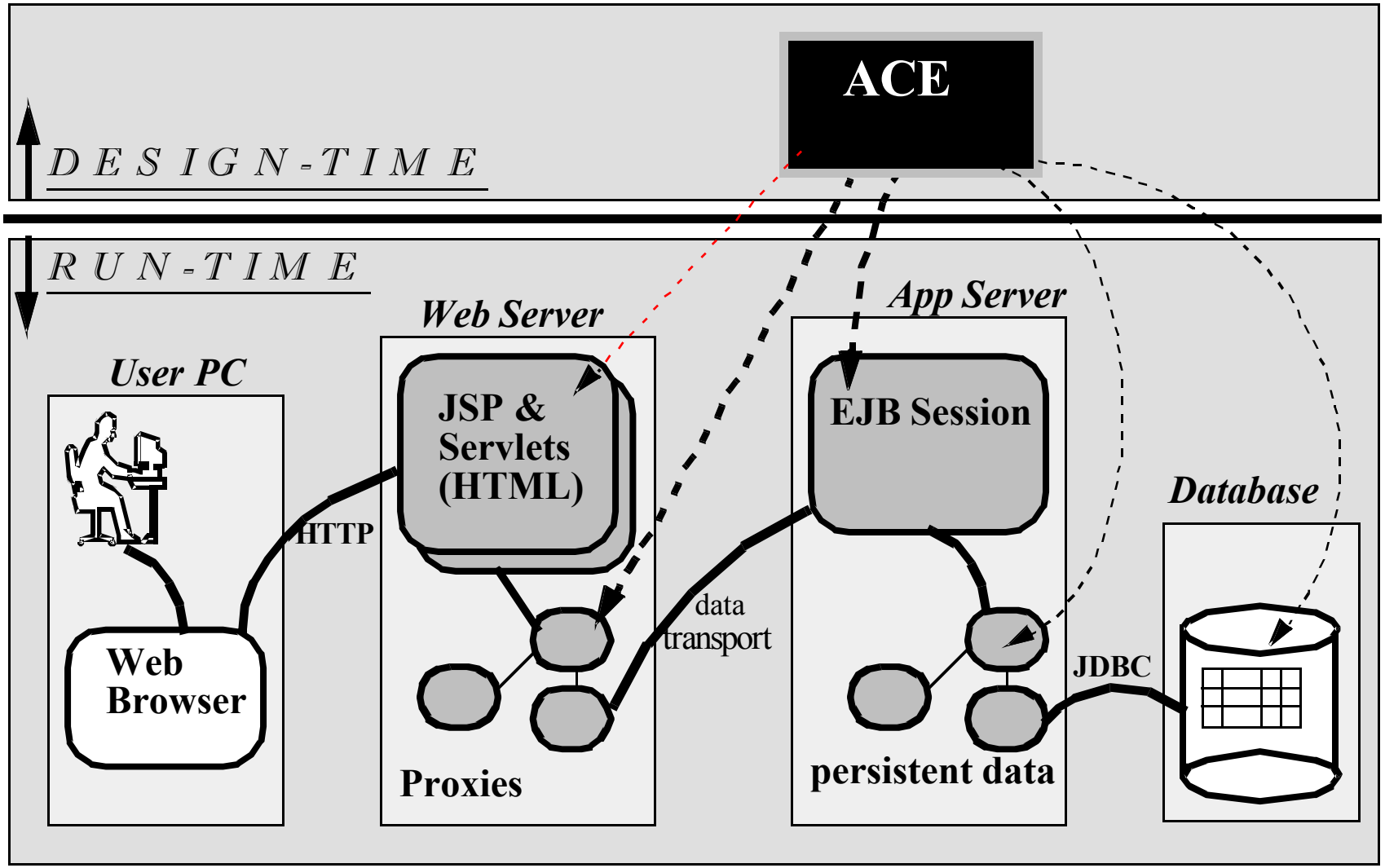
```
• initial state {
   entry { customerName = ""; }
   usage {
      customerName:  C;
    }
    transition Start {
      switch (customerName)
            case "": goto new initial();
            default: goto new ChoosePO();
      }
   } // transition Start
 } // state initial

state ChoosePO()
 {
    local Collection of PurchaseOrder purchOrds;
    entry {
       purchOrds = PurchaseOrder.myOrders
                          (customerName);
   }
.....
```

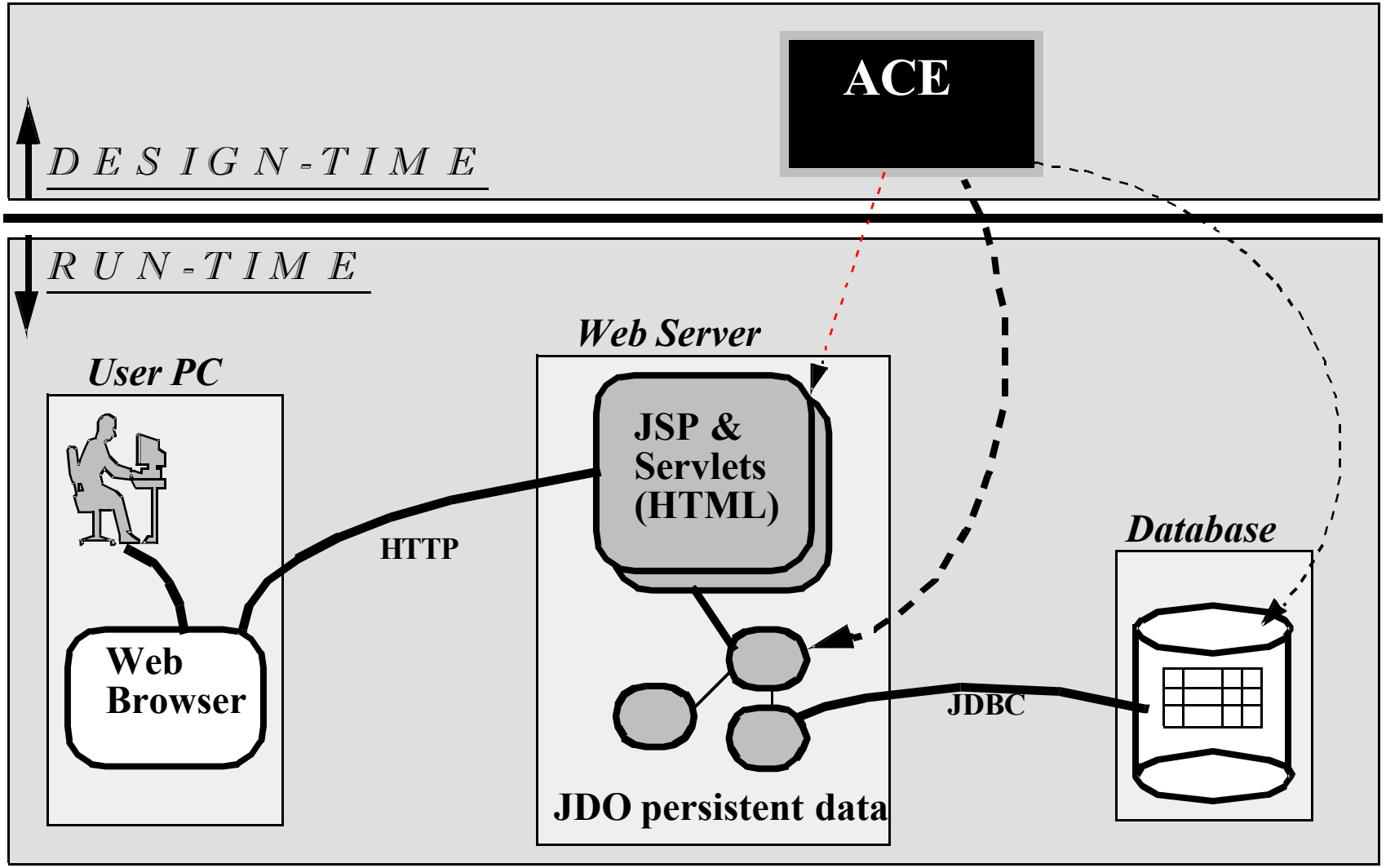
Implementation Generation

- Encapsulates Expertise & Automates Effort of Skilled App Programmer
  - hides details => critical to development productivity
- Optimizes Generated Code for High Performance & Scalability
- Provides Choice of Multiple Architectures, Technologies, & Vendors
  - change architecture ==> just regenerate code

Generate App as 3-Tier



Generate App as 2-Tier



Generate as Web Service

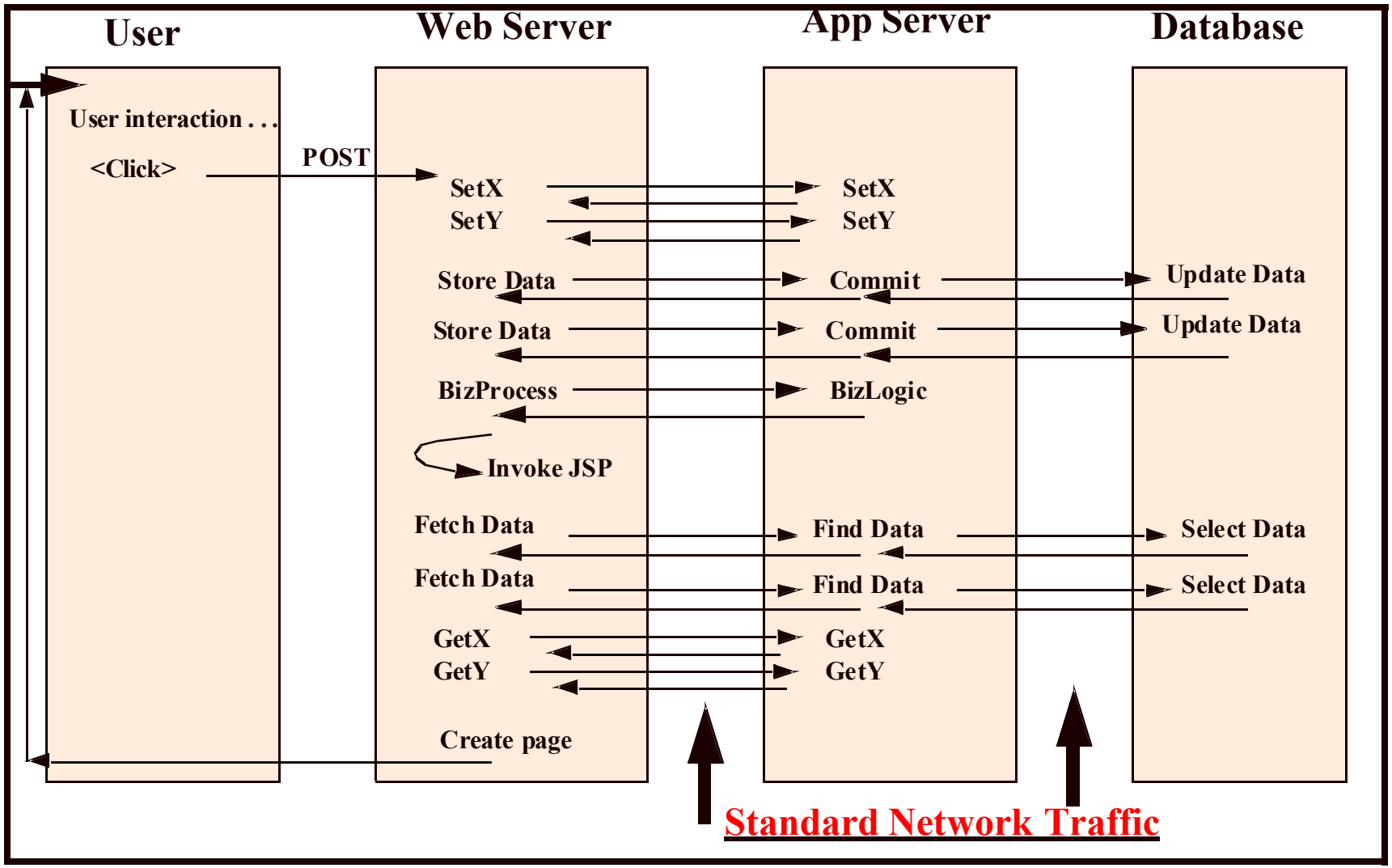
Each Application Task ==> Web Service

- Define Application Tasks through Tasks Designer
  - Import external Service definitions
- Auto-Generate SOAP XML API to invoke each Task as a Service
  - method for each Task user-operation
- Auto-Generate UDDI & WSDL registration of each Service
- Generate JSP Web Page to invoke each Service
- Offered as another architecture choice
  - existing apps will employ by simply re-generating code

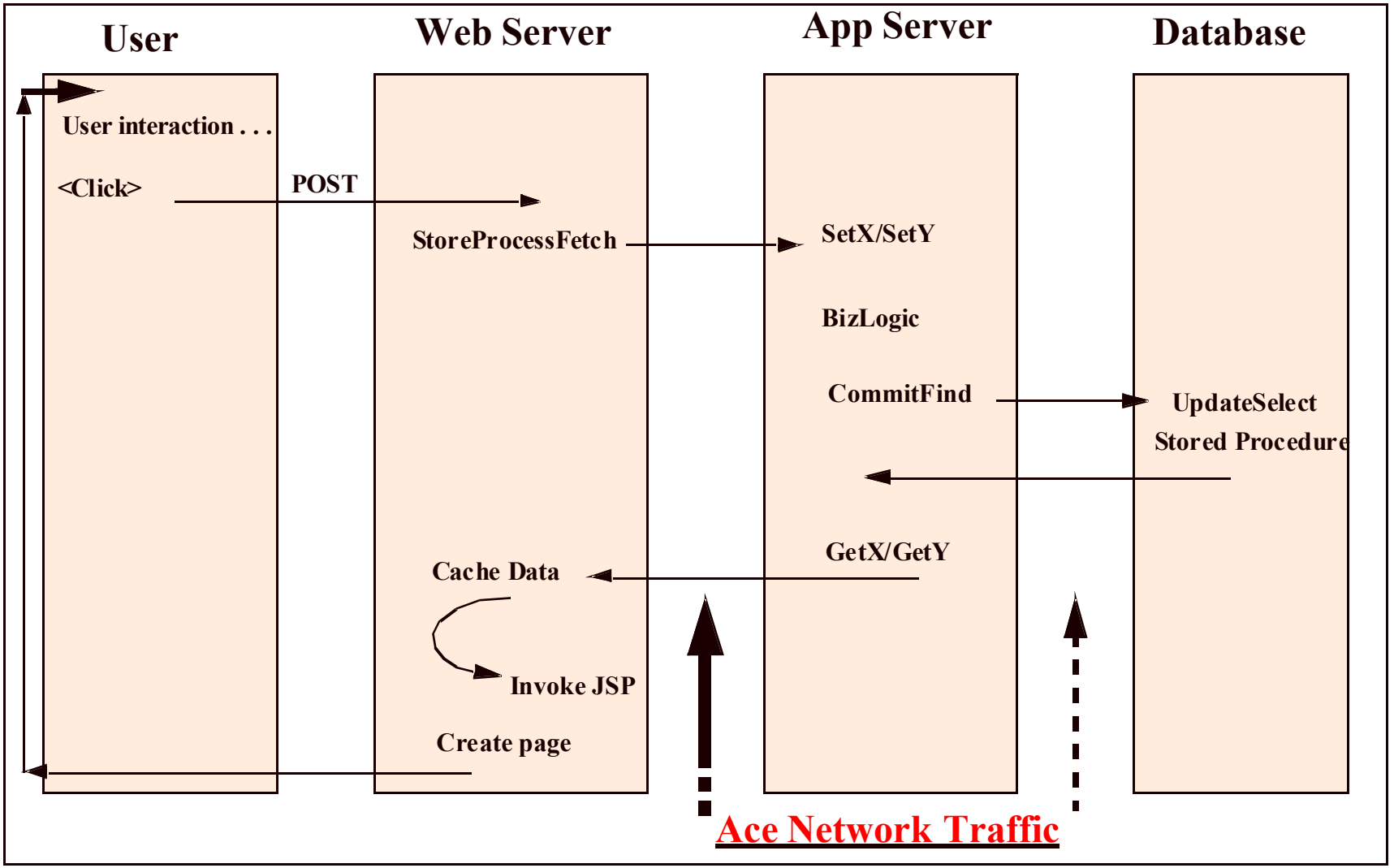
Performance Optimization

- Enterprise applications are inherently distributed
- Distribution efficiency is critical for performance
- Minimize network client/server round-trips
- Each application invokes different mix of operations
- Custom operation API needed to reduce invocations
  - e.g. custom EJB Session Bean
- Efficient custom API is too hard to design by hand
  - needs to be repeatedly performed as app evolves
- Specification Captures Complete Semantics of App
  - automatically optimize generated implementation for best performance

Conventional 3-Tier App



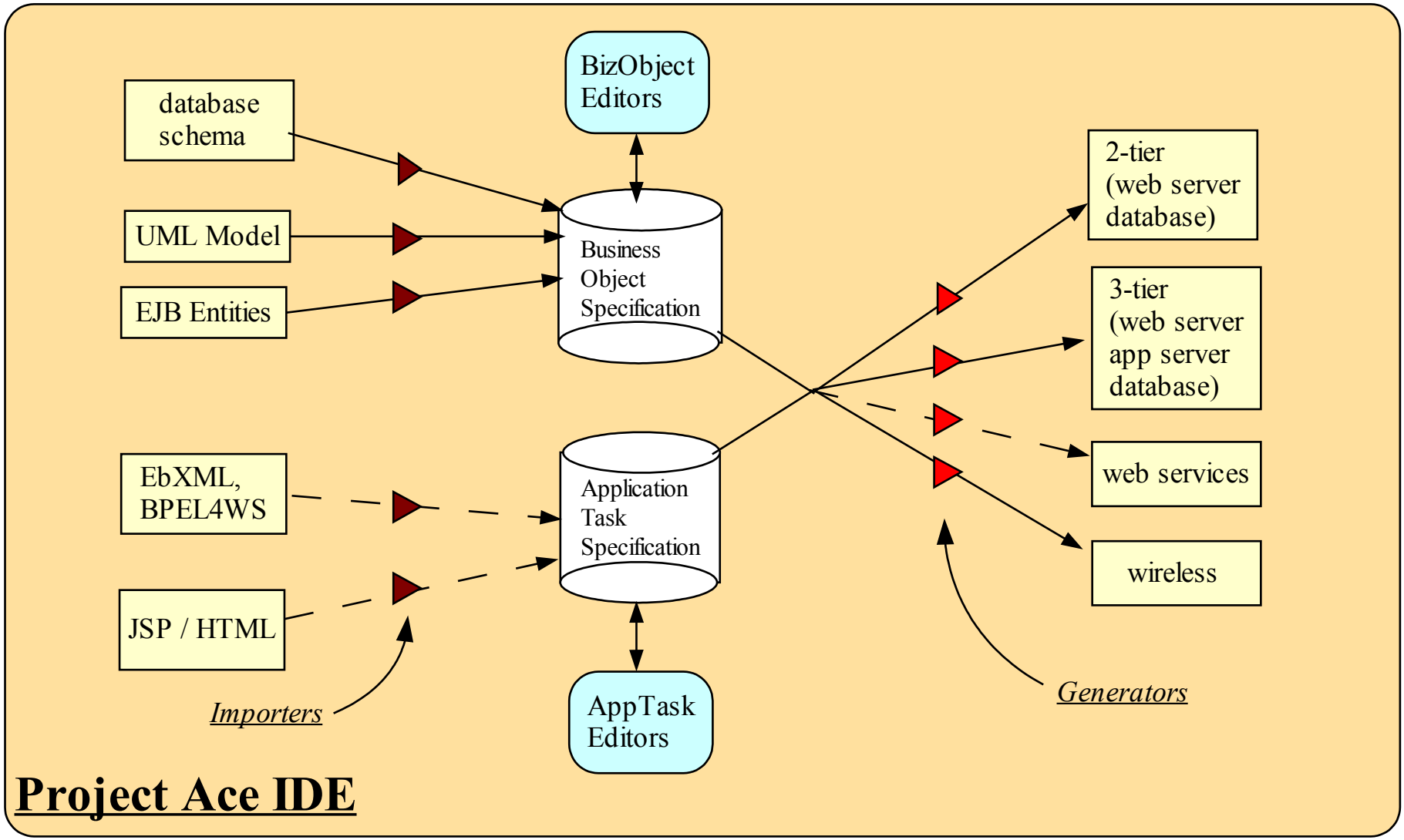
Ace Optimized 3-Tier App



Generated Code

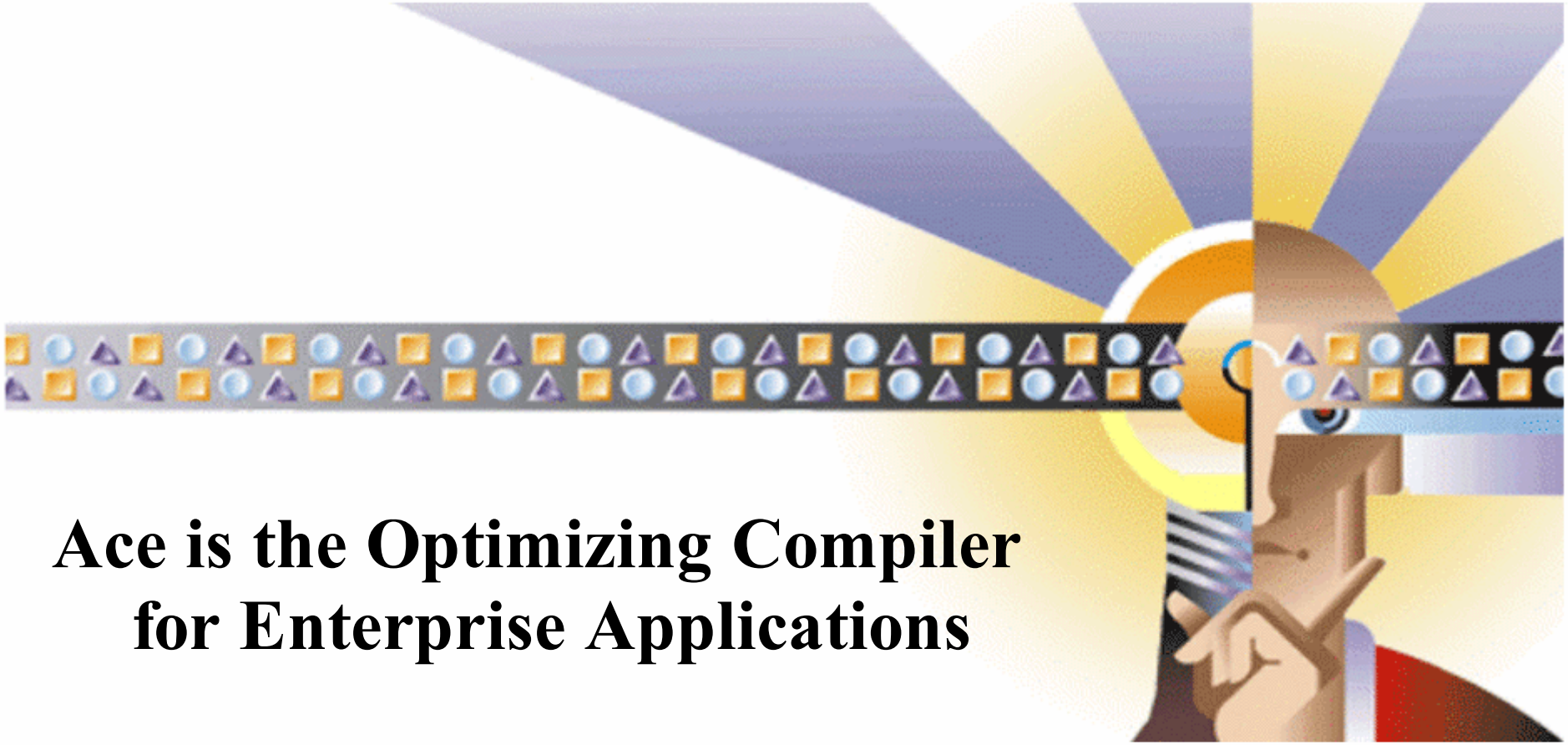
- "Normal" Source Code
  - just what a skilled programmer would write
- Invokes Standard API's
  - no proprietary execution engines (unlike 4GL's)
  - no proprietary subsystem API's
- Follows Good Programing Standards
  - well formatted and commented code
- Can be Maintained Independently
  - using normal development tools and practices

Specification/Generation Tool



Project Ace Results

- Java PetStore - “Reference App for EJB”
  - J2EE => 6 months & 15K lines code
  - .NET => ? months & 3.5K lines code
  - Ace => just 6 days & 250 lines code
- Accounts Receivable App – built by an Ace Developer Trial Customer
  - 14K lines of Ace textual spec
  - 320K lines generated code
  - 2 people in 8 weeks (4K lines per person per day)



Ace is the Optimizing Compiler for Enterprise Applications

Project Ace

http://research.sun.com/projects/ace/

Ace-Info@sun.com

Sun microsystems

We make the net work.