Agenda

- Introductions and topic review
- State of the Forms landscape
- Your options
  - Upgrade? Modernize? Which is right for you?
  - Upgrade or Modernize – points to consider
- Forms to Fusion Mapping
- ADF v11 pre-release – our findings
- Closing and Q&A
Vgo Software, Inc.

- Software and services to modernize Oracle Forms to JEE or upgrade
- 1 of 2 global partners certified by Oracle product development (only one in the US)
- 1st to migrate Forms to ADF v11
- 1st in the world to automate the conversion of character-based telnet forms
- Software and services based on 4 years of experience in conversions (25+ conversions)
- International customers and partners
- www.vgosoftware.com
Are Forms going somewhere? No.. maybe

- Oracle Forms 2008 de-support
  - v6.0.8.x **sustained support** ends in January 2008
  - 9.0.2.x **sustained support** ends in July 1 2008
  - 10gR2 extended support ends December 2011
  - 10gR3 extended support ends December 2011

- Forms continuing on until at least 2013
- Forms community is alive, well and strong
- But… Oracle is moving to ADF – you will too
  - Think about it… what is “Fusion”?
  - It’s important for Oracle’s continued growth strategy – it’s how they will integrate acquired products and those they’ve grown
  - What’s the “glue”? ADF, SOA Suite, BPEL, TopLink, a new (better) Enterprise Message Bus, etc..
Impact

What does this mean?

- If adopting ADF, JDeveloper will be the tool to use to do your work
- If you are not running Oracle Application Server (or Weblogic?), you will need to
- Get up to speed now – prepare for change
- Oracle Applications users – Apps are being migrated to ADF now; your custom extensions will have to move
- Resources – where will all the Forms developers be in 2012 (my guess – same beach as the Cobol developers)
Options

Upgrade to Web Forms

- **Pro’s:** Relatively easy, in-expensive and “safe”
- **Con’s:** Resource issues, true interoperability with other enterprise applications, 2012, 2013…?

Re-engineer to ADF

- **Pro’s:** Convert your business logic, create a thin-client architecture, faster than re-write, less expensive than re-write, consolidate redundant objects and clean out the code, look for opportunity to apply process re-design, think of Forms/ADF integration/coexistence points
- **Con’s:** Still some manual work, more expensive than upgrade, requires Java programming skills, watch out for “blind” conversions, gauge maintainability

Re-write using ADF

- **Pro’s:** Exactly what you want, generation capabilities in JDeveloper
- **Con’s:** Start with green-field design, re-application/creation of business rules, new test cases – time and cost, resource issues (i.e. training to the point of productivity)
Which Option is Right for You?

**Upgrade Forms when…**
- The application cannot change at all
- Cost is a huge issue
- You need to stay supported (6i users)

**Evolve to ADF when you require …**
- Greater ability to integrate with other applications
- Greater cross-platform support
- A “pure” thin client architecture
- Open architecture based on Java technologies

**Re-write when..**
- Business processes need to dramatically change
- You have unlimited budget
Upgrade Options

- Forms Builder
  - Open, compile and save the Form Module in the Forms Builder
  - Instant feedback
  - Least time efficient

- Batch Compiler
  - Create a batch script (DOS or shell) with upgrade flag on
  - Creates FMX’s for all the FMB’s
  - Check the .ERR file constantly (automated or manual)
  - Resolve errors (in your original Forms) and rerun

- JDAlP
  - Programmatically manipulate Forms
  - Need to have Java expertise – written in Java

- Migration Assistant
  - Command line interface ifplsqlconv90.exe (GUI available on OTN)
  - Makes changes if possible
Evolution Considerations

1. **Future state direction: SOA, EAI, etc**

2. **Organizational preparedness**
   - Staff resources
   - Impact to business: prepared for change
   - Testing and QA maturity levels

3. **Tie in functional changes**
   - Any impeded by the old app can be brought in
   - Add value to the project
   - Strategically orient the conversion to add functionality – if it’s planned right, it’s doable

4. **A dose of realism…**
   - This is not an easy move
   - It requires time and attention
Pre-Conversion Thoughts

Prior to any conversion, effort should be extended to:

- Consider strategic direction and business value
- Domain modeling
- Emphasize re-use and process
- Make ‘High-Leverage’ Enhancements
- Develop your “Conversion Framework”
- Think ahead – where and how does this tie in?
Typical Architectural View

To the Nth Degree

Redundant logic, process, data, tight integration with DB’s
# Forms to Fusion Mapping

<table>
<thead>
<tr>
<th>Legacy Environment</th>
<th>Fusion or ADF Mapping</th>
<th>Complexity</th>
<th>When?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forms</td>
<td>ADF v11 TR3</td>
<td>Mid/High</td>
<td>Specific to app</td>
</tr>
<tr>
<td>LOVs</td>
<td>Read-only View Objects</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query-based blocks</td>
<td>View Objects</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tables (DB)</td>
<td>Entity Objects</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transactions</td>
<td>Task Flows</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows &amp; Canvases</td>
<td>JSPX pages &amp; PanelGroups, resp.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB PL/SQL</td>
<td>Leave in DB</td>
<td>Simple</td>
<td>Specific to app</td>
</tr>
<tr>
<td></td>
<td>Convert to ADF</td>
<td>Medium/Low</td>
<td>Better performance, broader use</td>
</tr>
<tr>
<td></td>
<td>Web Service (S OA suite)</td>
<td>Medium/Low</td>
<td>External use</td>
</tr>
<tr>
<td>File processing</td>
<td>BPEL</td>
<td>High</td>
<td>External parties involved</td>
</tr>
<tr>
<td></td>
<td>ADF</td>
<td>Medium</td>
<td>Re-use in silo</td>
</tr>
<tr>
<td></td>
<td>Rule Repository</td>
<td>Low</td>
<td>Cataloging</td>
</tr>
</tbody>
</table>
Why ADF?

- **ADF Business Components**
  - Data Caching
  - Transaction Handling
- **ADF Rich Faces**
  - Ajax-enabled Components
  - JSF-Based
- **JDeveloper**
  - Lots of Wizards
  - Less Coding
Forms -> ADF

- Model Layer
  - Tables
    - Entity Objects
    - View Objects
    - View Links / Associations
  - Query-Based Blocks -> View Objects
  - LOVs -> read-only View Objects
Forms -> ADF

Forms == SQL
Very Simple
A Little Accomplishes a Lot
ADF == ADF BC
Multiple Layers
Separation of Interests
Forms -> ADF

- ViewController Layer
  - Forms -> Taskflows
  - Windows & Canvases
    - Windows become separate JSPX pages
    - Canvases become mutable PanelGroups
      - Hide/show each PanelGroup as application navigates between canvases
What’s so groovy about

- Tech Preview for ADF 11.0.0.0 includes Groovy 1.0
- Groovy scripts can be used in any application layer
  - validation on the presentation layer
  - foreign key checks on the Entity
What Doesn’t Map?

- Client-Server != Web
  - Many aspects of a Forms application can’t be replicated exactly in the web world
    - WinAPI Calls
    - File-system or Registry access
    - Synchronize
  - Many aspects of a Forms application should be re-implemented
    - Declarative validation
    - LOVs
Summary

- Upgrade for safety and ease
- Evolve when positioning for strategic advantage
- Always consider your process in either type of project – leverage new architectural advantages
- ADF v11 will be the correct go-forward answer
- Think outside of the Forms silo – what is the value to the business and to the enterprise?
Closing & Q/A

Contact Info
Ernst Renner, ernstr@vgosoftware.com
Rob Nocera, rnocera@vgosoftware.com

Rob’s blog on ADF, Java and miscellaneous thoughts www.java-hair.com