Database / User Class and Composite

Best Practices or Best Compromise
Me: Kim Ginnerup

I have been working with Ingres / ABF and OpenROAD for a very long time. 😊

I vaguely remember when I started but it was back in the days where Ingres was version 5 and when OpenROAD was called Windows4GL
This session

• It is not about nice UI
• It is not about nice code
• It is about structure and design behind the scene
• No design is all wrong
• No design is all right
• Your choice of design is a compromise
• The Frequent Flyer Database and Application
The Design Paradox

• Highly generic with a high level of abstraction
  • Difficult
  • Expensive
  • Deteriorate by design
  • You lose the objective

• Domain specific with a low level of abstraction
  • Easy
  • Inexpensive (at first)
  • Deteriorate by lack of design
  • The first 80% is done fairly easy. The last 20% is impossible
Design is a Compromise

• Define Your System Quality attributes
  • Sub-second response time
  • Maintainability
  • Customizable
  • Reliability
  • Readability
  • Learnability
  • Scalability
  • And many, many more

• You think you need them all
  • You don’t. But it’s an easy answer

• I like the design and code to be as readable as possible
• I like a high degree of decoupling
Your framework

- OpenROAD is a framework
- Make your extensions as simple as possible, but not simpler
- OpenROAD is driving your application, use it.
- Do not be a slave of your own design
  - Be flexible
- You know so much more about the domain tomorrow
  - Which is also correct tomorrow
Example 0: No Class approach

- Easy to use
- Fast development
- Easy to understand
- It is not a bad design if
  - it is a small system
  - It is static
  - Application Server is not needed
No Classes

- Total
  - 1 Frame
  - 4 Tables
  - 4 Foreign keys
  - 30 Statements
  - 83 Lines of code
- But
  - 2 identical selects on Airport
  - Only visible fields

Nothing is reusable
Example 1: Class model from frequent flyer (ff)

<table>
<thead>
<tr>
<th>Name</th>
<th>Remark</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Airline</td>
<td></td>
<td>Saved</td>
</tr>
<tr>
<td>Airport</td>
<td></td>
<td>Saved</td>
</tr>
<tr>
<td>Demo</td>
<td>Demo 1: Displays routes</td>
<td>Saved</td>
</tr>
<tr>
<td>Route</td>
<td></td>
<td>Saved</td>
</tr>
<tr>
<td>TableObject</td>
<td>Table Superclass</td>
<td>Saved</td>
</tr>
</tbody>
</table>

- 3 Data classes from ff added
  - Airport
  - Airline
  - Route
- Route holds the model
- Added in this example
  - Route has a collection of itself
  - TableObject is Superclass for Route
- Classes added to composites
A Class can be a collection of itself

- Easy to use
- Single Class
- Application Server doesn’t work
  - Collection of itself
  - Recursion is not allowed
    - JSON-RPC solves this
- You cannot read data into the list from the Class initialize block
  - Collection of itself
- It is not a bad design
  - It is a choice
A Class can reference other Classes

- Easy to use
- Model defined in Class
- Application Server will not work
  - nested cyclic references
  - Airport has an array of Route
- Model can
  - be extended without changing UI
  - not be changed without changing dependent UI
- UI layout is dictated by the data model
- It is not a bad design
  - It is a choice
Example 1

- Total
  - 1 Frame
- 4 Classes
- 4 Tables
- 4 Foreign keys
- 31 Statements
- 90 Lines of code
Example 2: Move collection to another class

- Easy to use
- Dual Class
  - 2 classes if Collection is needed
- Application Server will not work
  - nested cyclic references
  - Too much data may be passed
- UI layout is dictated by the data model
- It is not a bad design
  - It is a choice
Example 2

- Total
  - 1 Frame
  - 5 Classes
  - 4 Tables
  - 4 Foreign keys
  - 23 Statements
  - 83 Lines of code

```
Example 2

• Total
  • 1 Frame
  • 5 Classes
  • 4 Tables
  • 4 Foreign keys
  • 23 Statements
  • 83 Lines of code

```

```
Example 2

- Total
  - 1 Frame
  - 5 Classes
  - 4 Tables
  - 4 Foreign keys
  - 23 Statements
  - 83 Lines of code

```

```
Example 2

- Total
  - 1 Frame
  - 5 Classes
  - 4 Tables
  - 4 Foreign keys
  - 23 Statements
  - 83 Lines of code

```
Example 3: Move model from Class to Frame

• Makes it easier to layout the form
  • Fewer dependencies between data model and frame
• A model is built everytime a frame is created
  • So why build it in the classes as well
Move model from Class to Frame

- Easy to use
- Dual Class
  - 2 classes if Collection is needed
- Application Server will work
  - Less data transferred
  - Less chance for cyclic references
- It is not a bad design
  - It is a choice
Example 3

- Total
  - 1 Frame
  - 5 Classes
  - 4 Tables
  - 4 Foreign keys
  - 22 Statements
  - 79 Lines of code

### Routes.ToArray

<table>
<thead>
<tr>
<th>Airline</th>
<th>Flight num</th>
<th>Depart from</th>
<th>Arrive to</th>
<th>Depart at</th>
<th>Arrive at</th>
<th>Arrive off</th>
<th>Flight day</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAL</td>
<td>4625</td>
<td>ANW</td>
<td>AOJ</td>
<td>11-Oct-2006 10:10:00</td>
<td>11-Oct-2006 14:22:00</td>
<td></td>
<td>1234567</td>
</tr>
<tr>
<td>AAL</td>
<td>4626</td>
<td>AOJ</td>
<td>ANW</td>
<td>11-Oct-2006 03:08:00</td>
<td>11-Oct-2006 10:39:00</td>
<td></td>
<td>123456</td>
</tr>
<tr>
<td>AAL</td>
<td>4627</td>
<td>ANW</td>
<td>ARN</td>
<td>11-Oct-2006 06:11:00</td>
<td>11-Oct-2006 12:48:00</td>
<td></td>
<td>123456</td>
</tr>
<tr>
<td>AAL</td>
<td>4628</td>
<td>ANW</td>
<td>ANW</td>
<td>11-Oct-2006 11:15:00</td>
<td>11-Oct-2006 21:03:00</td>
<td></td>
<td>123456</td>
</tr>
<tr>
<td>AAL</td>
<td>4675</td>
<td>ANW</td>
<td>LAX</td>
<td>11-Oct-2006 07:06:00</td>
<td>11-Oct-2006 11:33:00</td>
<td></td>
<td>123456</td>
</tr>
<tr>
<td>AAL</td>
<td>4676</td>
<td>ANW</td>
<td>ANW</td>
<td>11-Oct-2006 11:42:00</td>
<td>11-Oct-2006 11:42:00</td>
<td></td>
<td>123456</td>
</tr>
</tbody>
</table>
Example 4

• This time all state has been removed from the Routes collection
  • Super Class: _Collection is no longer needed
• State is now held in the frame
• The Datamodel has been removed from all classes
  • Even current Route is removed from Routes
• The model is now built in the User Frame
  • Why not, you build it anyway
  • So why do it twice
• Requires more code in the frame but less code in the Classes
  • Less state
  • Fewer dependencies
Example 4

- Total
  - 1 Frame
  - 4 Classes
  - 4 Tables
  - 4 Foreign keys
  - 20 Statements
  - 70 Lines of code
### Status so far

<table>
<thead>
<tr>
<th></th>
<th>No Class</th>
<th>Example 1 One Class</th>
<th>Example 2 Collection Class</th>
<th>Example 3 Move model</th>
<th>Example 4 State in frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of classes</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Statements</td>
<td>30</td>
<td>31</td>
<td>23</td>
<td>22</td>
<td>20</td>
</tr>
<tr>
<td>Lines of Code</td>
<td>83</td>
<td>90</td>
<td>83</td>
<td>79</td>
<td>70</td>
</tr>
<tr>
<td>Attributes</td>
<td>0</td>
<td>28</td>
<td>31</td>
<td>27</td>
<td>22</td>
</tr>
<tr>
<td>Methods</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>3</td>
</tr>
</tbody>
</table>
What we have accomplished

• Inter class dependencies has been reduced
  • Without an increase in complexity
• State is held where state is changed (in the frame)
• Made the model more dynamic
• Created a stateless DB Layer
• We do not have 1 Table equals 1 Class
  • We access the database using SQL the right way
• Statements and LoC is reduced
  • Even compared to the No-Class solution (surprising!)
Questions

We have one more thing
Let’s go 3-tier

• Move everything but the frame into the server
  • A bit of an easy trick in this example as we do not have any business logic
• We go the stateless route
  • I am not a fan of statefull server interfaces
  • Let us use Connect->Call->Disconnect
• What we need:
  • A proxy layer at the client side
    • Classes or 4GL Procedures
  • A Service layer at the Server side
    • Procedures
      • JSON-RPC will reduce a lot
• Built as one application
Example 5: Application Server

- Total
  - 1 Frame
  - 1 Ghostframe
  - 5 Classes
  - 3 4GL Procedures
  - 4 Tables
  - 4 Foreign keys
  - 41 Statements
  - 124 Lines of code

- Surprising increase in code
  - Proxy methods
  - Server 4GL Procedures

- JSON-RPC will reduce this
  - All 4GL Procedures will be reduced to
    - 1 generic 4GL Procedure
    - A json config file
Code added to enable Server

**Data Class**

Method `RemoteGet`

```
Method RemoteGet(
    IcaoCode = varchar(3) not null;
)=
{
    CurRemoteServer.initiate(
        Image = '$_applicationName',
        Type = RP_LOCAL,
        Routing = 'HTTP',
        Location = 'localhost');

    CurRemoteServer.call4GL('airline_get',
        IcaoCode = IcaoCode,
        curObject = Byref(CurObject));*

    CurRemoteServer.Release();
}
```

**Service Call Procedure**

Procedure `airline_get`

```
Procedure airline_get(
    IcaoCode = varchar(3) not null,
    CurObject = Airline
)=
{
    CurObject.Get(icaoCode = IcaoCode);
}
```

**Server Starting Component**

`initialize()` =

```
initialize={
    $_curfld.get();
}

on ChildEntry=
{
    $_curfldsetCurrent(row = Field($_curfld.list).currow);
    Field($_curFld).updField();
}
Frame Source Code 3

```javascript
initialize={
    $_curfldld.get();
}
on ChildEntry={
    $_curfldld.SetCurrent(row = Field($_curfldld.list).currow);
    AirlineUI.get(icaoCode = $_curfldld.obj.Airline#);
    DepartFrom.get(IataCode = $_curfldld.obj.DepartFrom#);
    ArriveTo.get(IataCode = $_curfldld.obj.ArriveTo#);
    CurFrame.TopForm.updField();
}
```
initialize={
    $_curfld.get();
    Rows = $_curfld.list.LastRow;
}

on ChildEntry=
{
    CurRow# = Field($_curfld.list).currow;
    RouteUI = $_curfld.List[CurRow#];
    AirlineUI.get(icaoCode = RouteUI.Airline#);
    DepartFrom.get(IataCode = RouteUI.DepartFrom#);
    ArriveTo.get(IataCode = RouteUI.ArriveTo#);
    Curframe.topform.updField();
}
initialize={
    $_curfld.get();
    Rows = $_curfld.list.AllRows;
}

on ChildEntry=
{
    CurRow# = Field($_curfld.list).currow;
    RouteUI = $_curfld.List[CurRow#];
    AirlineUI.RemoteGet(icaoCode = RouteUI.Airline#);
    DepartFrom.RemoteGet(IataCode = RouteUI.DependFrom#);
    ArriveTo.RemoteGet(IataCode = RouteUI.ArriveTo#);
    Curframe.topform.updField();
}