Consultingwerk
software architecture and development

The OERA Maturity Model or, why ORMs can be good for you

Peter Judge, Consultingwerk

Peter Judge

- Senior Architect at Consultingwerk
- Writing 4GL since 1996, working on a variety of frameworks and applications. More recently have worked on a lot of integration-y stuff: Authentication Gateway, HTTP Client, Web Handlers. Dabble in PASOE migrations.
- Active participator in Progress communities, PUGs and other events

Consultingwerk Software Services Ltd.

- Independent IT consulting organization
- Focusing on **OpenEdge** and **related technology**
- Located in Cologne, Germany, subsidiaries in UK, USA and Romania
- Customers in Europe, North America, Australia and South Africa
- Vendor of developer tools and consulting services
- Specialized in GUI for .NET, Angular, OO, Software Architecture, Application Integration
- Experts in OpenEdge Application Modernization

Services Portfolio, Progress Software

- OpenEdge (ABL, Developer Tools, Database, PASOE, …)
- Telerik DevCraft (.NET, Kendo UI, Angular, …), Telerik Reporting
- OpenEdge UltraControls (Infragistics .NET)
- Telerik Sitefinity CMS (incl. integration with OpenEdge applications)
- Kinvey Plattform, NativeScript
- Corticon BRMS
- WhatsUp Gold infrastructure-, network- and application monitoring
- Kemp Loadmaster
- …

Services Portfolio, related products

- Protop Database Monitoring
- Combit List & Label
- Web frameworks, e.g. Angular
- .NET
- Java
- ElasticSearch, Lucene
- Amazon AWS, Azure
- DevOps, Docker, Jenkins, ANT, Gradle, JIRA, …
- …

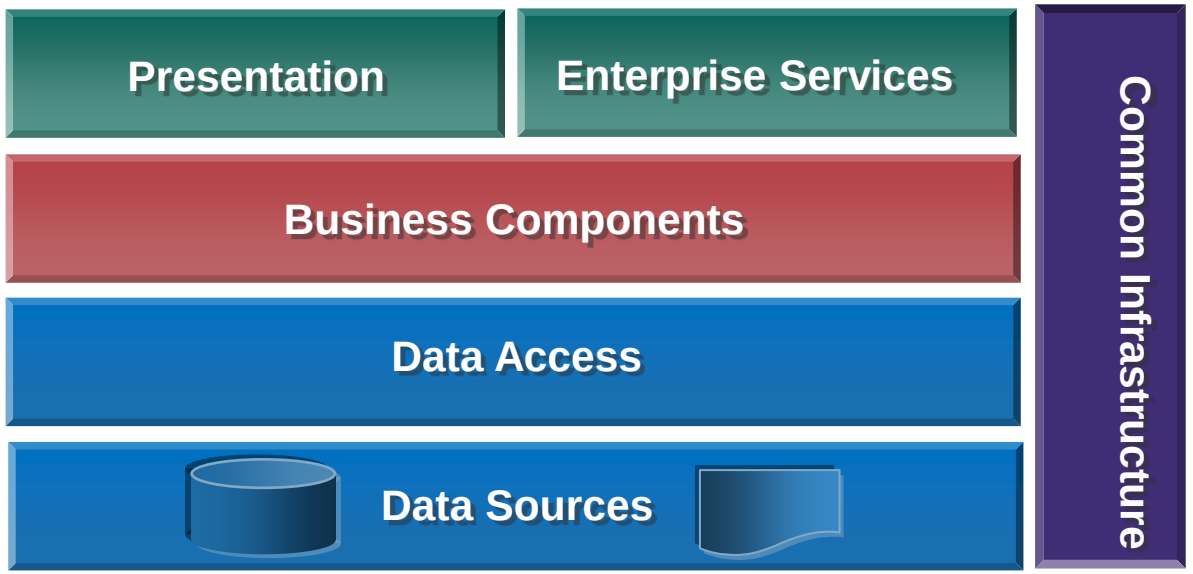
Agenda

- The OpenEdge Reference Architecture
- Extensions
- Tasks and launching
- Snippets and templates

The OpenEdge Reference Architecture (OERA)

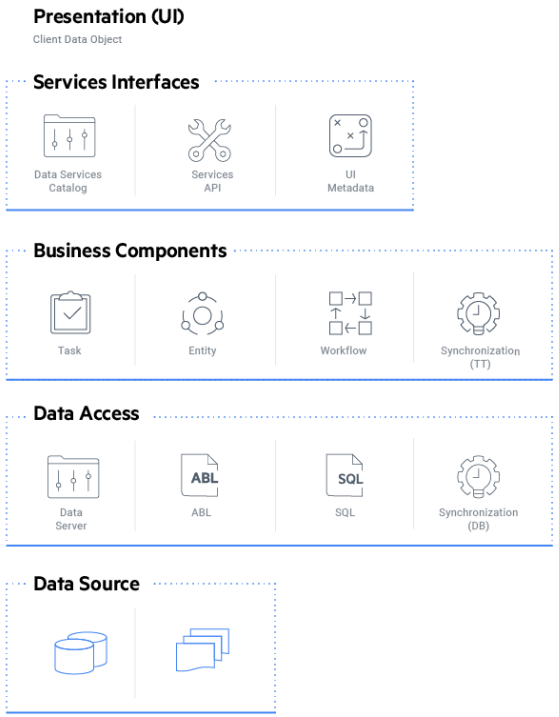
> *The OpenEdge Reference Architecture (OERA) defines the general functional categories of components that comprise an application. It can be used as a high-level blueprint for developing OpenEdge service-oriented business applications."*

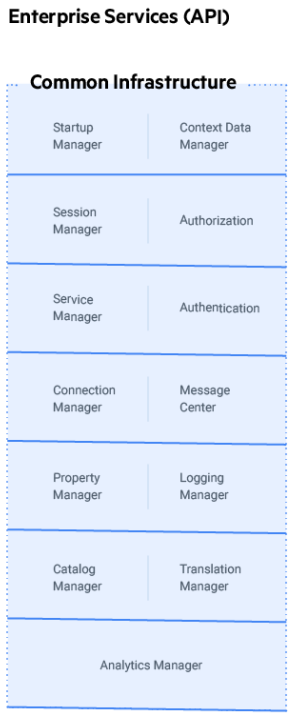
- A guide only – not prescriptive
- Implementations vary, sometimes wildly

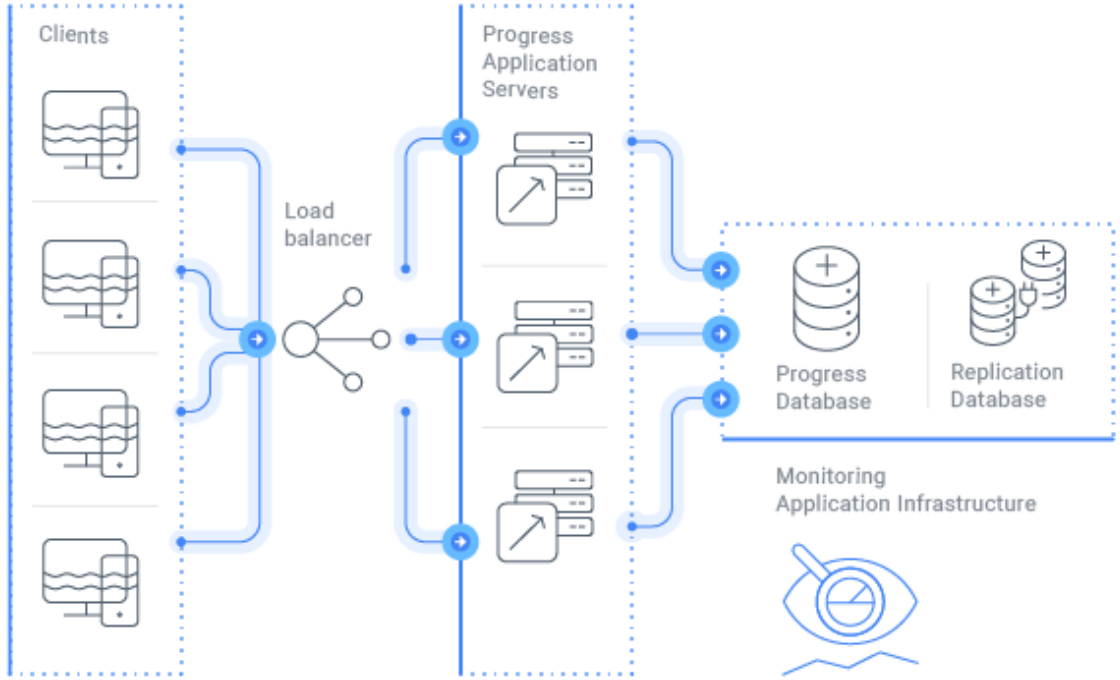


https://community.progress.com/s/question/0D54Q0000819wkqSAA/introduction-to-the-openedge-reference-architecture

Updated as the OpenEdge Application Architecture …





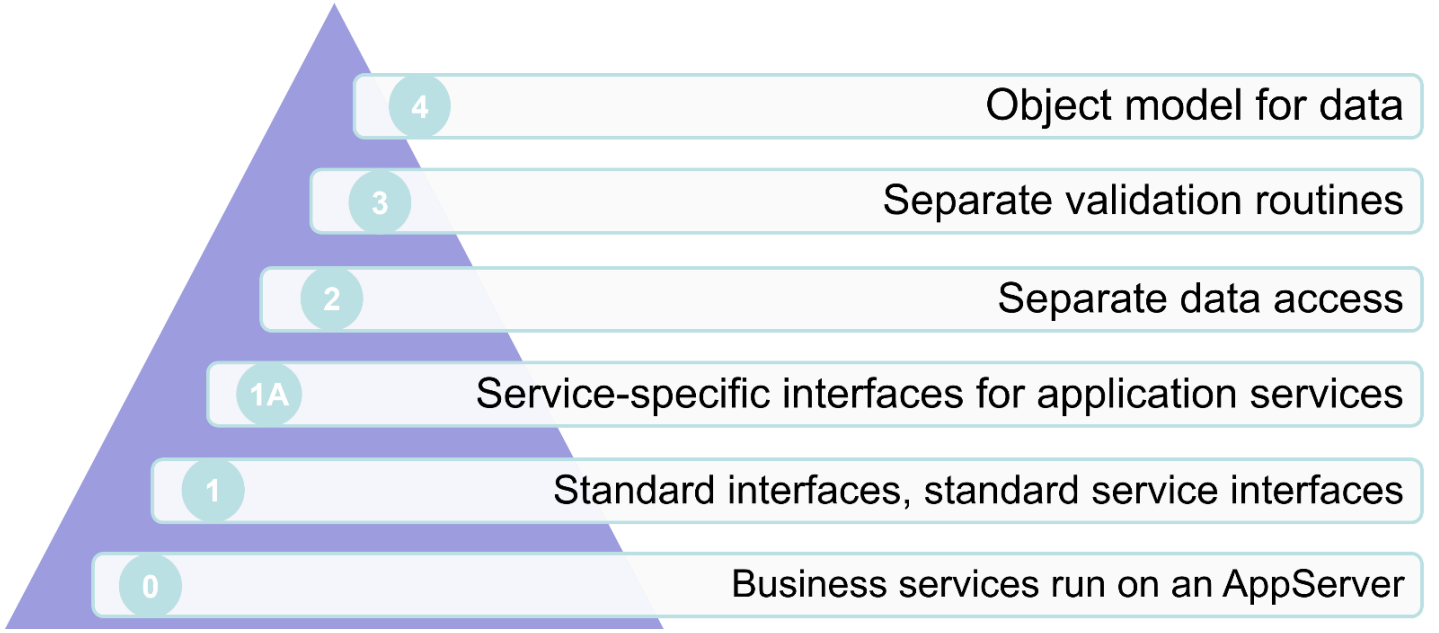


https://docs.progress.com/bundle/openedge-modernize-guide-122/page/OpenEdge-application-design.html

https://docs.progress.com/bundle/openedge-modernize-guide-122/page/OpenEdge-deployment-design.html

The OERA Maturity Model

- An opinionated approach implement the principles of the OERA, to help future-proofing of an application
  - PASOE – or something compatible – will be around for a decade or more
  - "Classic" AppServer was released in 1999, will be "Retired" in 2025
- Targeted at application modernization
- Focuses on the backend of a business application



Level 0: Business Services running on an AppServer

- Business services exist and run on an AppServer
  - Written in OOABL or procedural ABL
  - Have their own API (method/function names and/or parameter definitions)
  - Data structures are ProDatasets, temp-tables, OOABL objects or primitive parameters
  - Clients are ABL, SOAP and/or RESTful

✓ Having a service interface qualifies the application as OERA compliant
  - Often missing for ABL clients

Guiding principles: separation of concerns

- Follow the separation of concerns principle aka the Single Responsibility Principle (SRP)
  - Developers group like functions and handle them mentally in the same way
  - Multiple exceptions to these groups is a sign that a new concern/responsibility exists and should be 'captured' in common code
  - Can make code reuse harder
- Reusing existing code allows potentially faster migration
  - Existing code may only be partial duplicates, increasing developer confusion
- Need to balance migration speed and risk vs. increased maintainability

Guiding principles: OOABL

- ✓ Use OOABL wherever possible
  - The compiler keeps developers (mostly) honest

- Combined with the SRP, we get many smaller programs
  - ✓ A more maintainable codebase, especially with automation
  - ✓ Fewer sprawling “God programs”
  - ✓ Changes are less likely to have large and unforeseen impacts

Level 1: Standard interfaces

- Business services must have a single responsibility
  - E.g. putting a customer on hold should not update the order data directly
  - Abstract responsibility into standardized interfaces

- Standardized interfaces can be for single services (Business Entities), or more complex services (Business Tasks)
  - Complex services are typically composed of a number of distinct services that have their own responsibilities

Service API

- Sample business service interface

```
USING Ccs.BusinessLogic.*.

INTERFACE Ccs.BusinessLogic.IBusinessEntity:
  METHOD PUBLIC VOID getDataset (
                              OUTPUT DATASET-HANDLE phDataset).
  METHOD PUBLIC IGetDataResponse getData (poRequest AS IGetDataRequest,
                              OUTPUT DATASET-HANDLE phDataset).
  METHOD PUBLIC IGetResultCountResponse getResultCount (
                              poRequest AS IGetDataRequest).


INTERFACE Ccs.BusinessLogic.IUpdatableBusinessEntity INHERITS IBusinessEntity:
  METHOD PUBLIC Progress.Lang.Object updateData (
                              INPUT-OUTPUT DATASET-HANDLE phDataset,
                              poUpdateDataRequest AS IUpdateDataRequest).
```

Service interface API

- One per client type
  - **ABL**
  - WEB
- Handles format translations, errors, authorization

```
block-level on error undo, throw.

using Ccs.BusinessLogic.*.

define input parameter pcBusinessEntity as character no-undo.
define input parameter poRequest as IGetDataRequest no-undo.
define output parameter dataset-handle phDataset.
define output parameter poResponse AS IGetDataResponse no-undo.

/* MAIN BLOCK */
define variable oBusinessEntity as IBusinessEntity no-undo.

oBusinessEntity = dynamic-new pcBusinessEntity().

poResponse = oBusinessEntity:getData (poRequest,
                                       output dataset-handle phDataset).


/* Errors are simply thrown back to the ABL client */

finally:
    delete object phDataset.
end finally.
```

Service interface API

- One per client type
  - ABL
  - **WEB**
- Handles format translations, errors, authorization

```
method override protected integer HandleGet ( input poWebRequest as IWebRequest ):
        assign oResponse             = new WebResponse(StatusCodeEnum:OK)
               oResponse:ContentType = 'application/json'.

        case entry(2, poWebRequest:PathInfo, "/"):
          when "Customer" then cBusinessEntity = "Application.BusinessLogic.CustomerBusinessEntity":u.
          otherwise undo, throw new AppError("Unknown business entity: " + entry(2, poWebRequest:PathInfo, "/"), 0).
        end case.

        oBusinessEntity = dynamic-new cBusinessEntity ().
        oGetDataRequest = this-object:BuildGetDataRequestFromQuery(poWebRequest).
        oGetDataResponse = oBusinessEntity:getData (oGetDataRequest, output dataset-handle hDataset).

        hDataset:write-json("JsonObject":u, oDatasetJson, false).
        oResponseJson:Add("meta", this-object:BuildJsonFromGetDataResponse(oGetDataResponse)).
        oResponseJson:Add("data", oDatasetJson).
        oResponse:Entity = oResponseJson.

        this-object:WriteResponse(oResponse).

        return 0.

        /* Errors must be processed and formatted before being returned to the client */
        catch oError as Progress.Lang.Error:
          oResponse:StatusCode = integer(StatusCodeEnum:InternalServerError).
          oResponseJson        = new JsonObject().
          oResponseJson:Add("error", oError:GetMessage(1)).
          this-object:WriteResponse(oResponse).
        end catch.
        finally:
          delete object hDataset.
        end finally.
```

Demo

- Service API
  - Business entities
- Service interfaces
  - Read.p
  - Handleget
  - etc

Level 2: Separate data access from business entities

- A Business Entity is considered to have a logical data model, represented as temp-tables and ProDataSets
- The data access component is responsible for
  - Mapping the logical model to the physical (eg temp-table field to db field)
  - Populating temp-table from physical storage
  - Committing temp-table data to physical storage
- The Data Access object is a black box to the Business Entity
  - For example, may have support for multi-tenancy
  - Allows for plug and play for data, including for test mocking
- OERA defines a distinct Data Source layer, but that's generally overkill

Level 3 : Separate validation routines

- A Business Entity MUST always perform its own validation (trust but verify)
- Validation may include ensuring that
  - A field has a value
  - Values are in a particular range
  - If one field has a value, another field has a related value (or empty)
- Validation logic will
  - Be reusable by multiple business services
  - Have its own set of interfaces
  - Follow the separation of concerns

Example: address validation routines

- Multiple Business Entities have one or more addresses
  - Orders: shipping and billing
  - Customer: postal and physical
- When updating an address, the same validation should apply in all cases
- We need one class whose concern is validating the Order
  - Called once per record in the Business Entity
- One class whose concern is validating an address
  - This will be called once for each address in an order, so once for shipping address and once for billing address

Demo

- Validation

Level 4 : Object model for data

- Temp-tables and ProDataSets are handle-based
  - Reference-passing can be tricky
  - Compile checks operate on the whole data structure: cannot define a temp-table as having multiple components
- Level 4 defines the temp-table and ProDataSet data models as a set of OOABL classes and interfaces
- Not replacements for the business services – these are typically used to populate the model objects … in some ways simply "syntactic sugar" for working with relational ABL data in temp-tables

Level 4 : Object model for data

- Implementation choices include classes that
  - Are wrappers around the complete data structures
  - Are wrappers around individual rows
  - Copy values from individual rows
  - Are collections of row-copy classes

- The ABL is comparatively powerful when working with relational structures, so the wrappers is our recommended option

Demo

- SCL dataset, table model

Conclusion

- The OERA Maturity Model provides opinionated approaches to implementing the server-side OERA components, and provides a mechanism to evaluate where further improvements can be found in an application
- Whitepaper at https://www.consultingwerk.com/news/blog/consultingwerk/2024/02/23/the-consultingwerk-oera-maturity-model

PUG Challenge 2024

- Europe: September 18th – 20th in Prague, CZ
- Americas: 29 Sept – 2 Oct, Waltham, MA

Questions


Consultingwerk
software architecture and development