Agenda

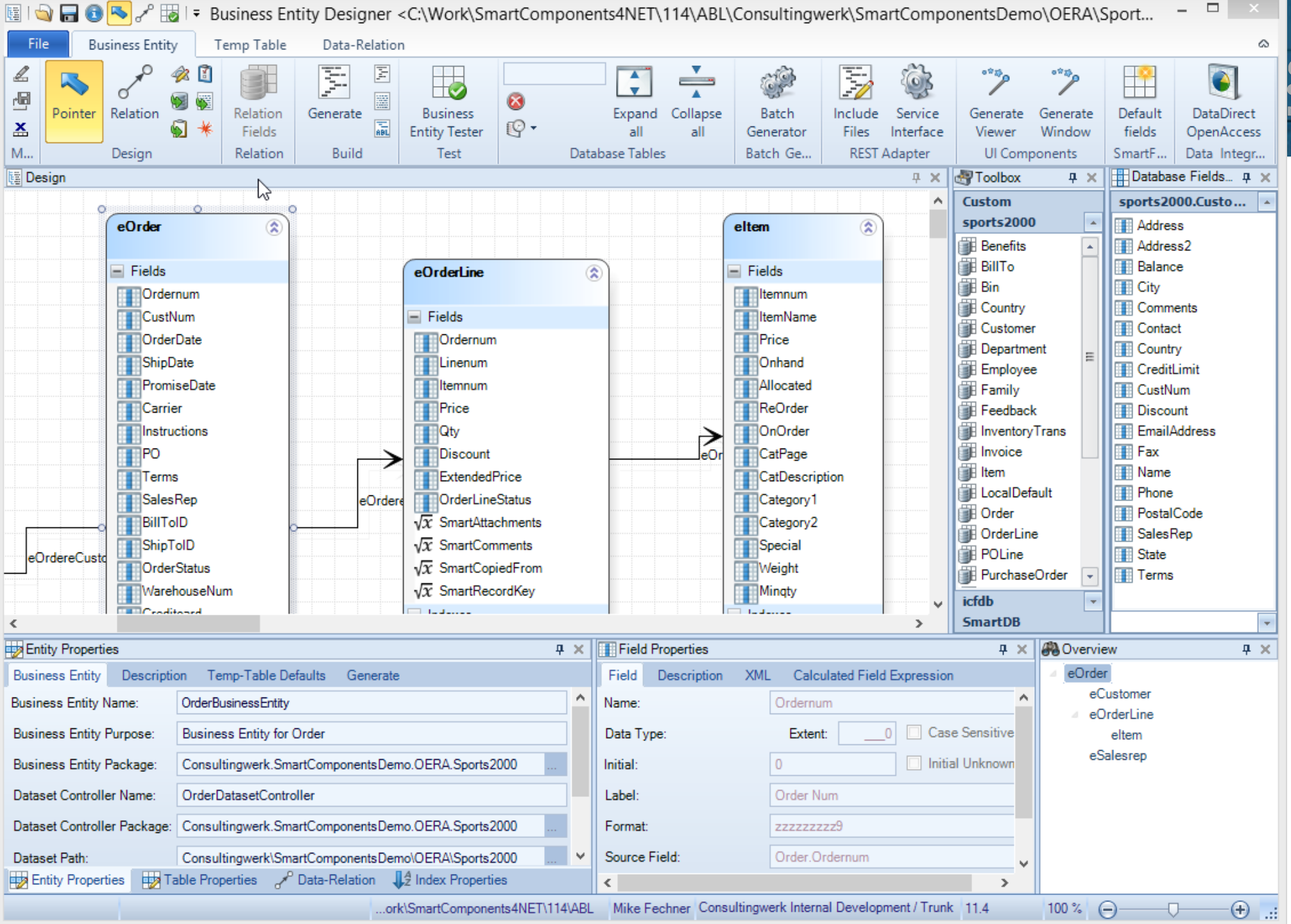
- **Introduction**
- Domain-driven Design
- OERA and CCS
- Implementing modules
- Implementing domain events
- Implementing value objects
- Implementing Entities

Consultingwerk

- Independent IT consulting organization
- Focusing on **OpenEdge** and **related technology**
- Located in Cologne, Germany, subsidiaries in UK 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

SmartComponent Library

- Developer Framework aimed to increase Developer productivity and flexibility
- Reduce or avoid repeating tasks
- Tools (code generation and round-trip dev.)
- Integration with various Progress tools (OpenEdge, Telerik, Kinvery, BPM, Corticon, DataDirect …)
- Architecture Framework and Application Framework
- Proud on our quality; frequent releases and almost no regression issues


Business Entity Designer <C:\Work\SmartComponents4NET\114\ABL\Consultingwerk\SmartComponentsDemo\OERA\Sport...
File
Business Entity
Temp Table
Data-Relation
Pointer
Relation
Design
Relation Fields
Relation
Generate
Build
Business Entity Tester
Test
Expand all
Collapse all
Database Tables
Batch Generator
Batch Ge...
Include Files
Service Interface
REST Adapter
Generate Viewer
Generate Window
UI Components
Default fields
SmartF...
DataDirect OpenAccess
Data Integr...
Design
eOrder
Fields
Ordernum
CustNum
OrderDate
ShipDate
PromiseDate
Carrier
Instructions
PO
Terms
SalesRep
BillToID
ShipToID
OrderStatus
WarehouseNum
eOrderLine
Fields
Ordernum
Linenum
Itemnum
Price
Qty
Discount
ExtendedPrice
OrderLineStatus
SmartAttachments
SmartComments
SmartCopiedFrom
SmartRecordKey
eItem
Fields
Itemnum
ItemName
Price
Onhand
Allocated
ReOrder
OnOrder
CatPage
CatDescription
Category1
Category2
Special
Weight
Minqty
Toolbox
Custom
sports2000
Benefits
BillTo
Bin
Country
Customer
Department
Employee
Family
Feedback
InventoryTrans
Invoice
Item
LocalDefault
Order
OrderLine
POLine
PurchaseOrder
icfdb
SmartDB
Database Fields...
sports2000.Custo...
Address
Address2
Balance
City
Comments
Contact
Country
CreditLimit
CustNum
Discount
EmailAddress
Fax
Name
Phone
PostalCode
SalesRep
State
Terms
Entity Properties
Business Entity
Description
Temp-Table Defaults
Generate
Business Entity Name:
OrderBusinessEntity
Business Entity Purpose:
Business Entity for Order
Business Entity Package:
Consultingwerk.SmartComponentsDemo.OERA.Sports2000
Dataset Controller Name:
OrderDatasetController
Dataset Controller Package:
Consultingwerk.SmartComponentsDemo.OERA.Sports2000
Dataset Path:
Consultingwerk\SmartComponentsDemo\OERA\Sports2000
Entity Properties
Table Properties
Data-Relation
Index Properties
Field Properties
Field
Description
XML
Calculated Field Expression
Name:
Ordernum
Data Type:
Extent:
Case Sensitive
Initial:
Initial Unknown
Label:
Order Num
Format:
zzzzzzzzz9
Source Field:
Order.Ordernum
Overview
eOrder
eCustomer
eOrderLine
eItem
eSalesrep
...ork\SmartComponents4NET\114\ABL
Mike Fechner
Consultingwerk Internal Development / Trunk
11.4
100 %

Framework Backend Architecture

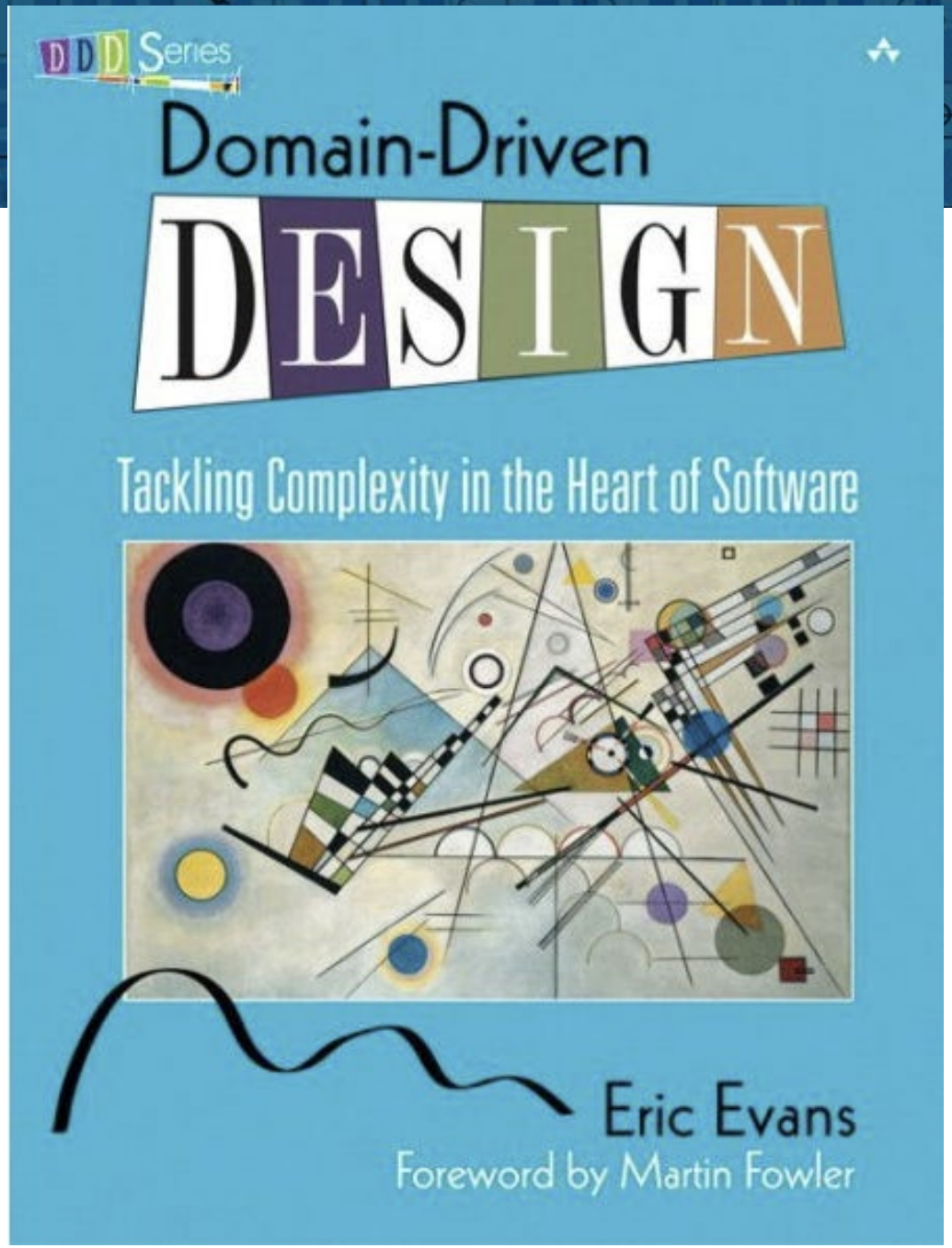
- Strong focus on modern application architecture
- OpenEdge Reference Architecture compliant
- Complies with the Common Component Specification (CCS)
- Business Entities, Data Access Objects are a key components
- Business Tasks, including support for scheduled and asynchronous processing
- ORM Elements und Domain Driven Design
- Common Infrastructure Components, Services

User-Interface Flexibility

- OpenEdge GUI for .NET
- Angular Web Applications (Telerik Kendo UI and JSDO)
- NativeScript
- Open standard interfaces (eg. RESTful, .NET, Java)
- Support for static user interfaces, repository based user interfaces and a combination of both

Agenda

- Introduction
- **Domain-driven Design**
- OERA and CCS
- Implementing modules
- Implementing domain events
- Implementing value objects
- Implementing Entities



Domain-driven Design

- Term coined by Eric Evans (working as a consultant on Domain-driven design) in his 2003 book
- Tackling complexity in the heart of software
- Introducing a software design methodology that allows domain-experts (business analysts) and developers to work together
- Design of applications that require complex domain knowledge
- Software design around the core domain of an application

Application design challenges

- Indirect communication between domain experts and developers
- Each focusing on its own terms
- Potentially separated by software designers and architects

- Domain expert may not care about database design or object inheritance

Domain-driven Design

- A domain specific project needs to leverage multiple realms of expertise
  - Domain specific expertise (key users and business analysts)
  - Design (software architecture) and Developer expertise (implementation)
- The Challenge
  - Need to enable communication between the two groups
  - Project organization can insulate the transmission of knowledge and retard the ideal evolution of a project

The Goal of Domain-driven design

- The Solution
  - enable and simplify the communication process and establish a methodology for making those communications more robust and efficient
  - primarily accomplished by developing a **ubiquitous language** and single model.
- Set design focus on application domain, not on implementation details
  - e.g. don't waste time talking about database tables and inheritance concepts
- Process supported by agile methodologies

Ubiquitous language

- "The vocabulary of that ubiquitous language includes **names of classes** and **prominent operations**. The language includes terms to discuss rules that have been made explicit in the model. It is supplemented with terms from high-level organizing principles imposed on the model. Finally, this language is enriched with the **names of patterns** the team commonly applies to the domain model"

  Eric Evans

Developers model

- Domain-driven design principles overlapping with model-driven design principles
- Developers are responsible for the model – “If developers don’t realize that changing code changes the model, then their refactoring will weaken the model rather than strengthen it.”, Eric Evans
- “With a MODEL-DRIVEN DESIGN, a portion of the code is an expression of the model; changing the code changes the model. Programmers are modelers, whether anyone likes it or not. So it is better to set up the project so that the programmers do good modeling work.”, Eric Evans

Elements of Domain model: *Entities*

- Entities are NOT the same as business entities in OERA/CCS
- Primary Objects of the domain model, e.g. Customer, Person
- Defined by an identity (e.g. a primary unique key, reference in DB), identity defined by reference, not by properties (two customers with the same name and address may still be two different customers)
- Changing properties of entity instance will not create a new entity
- May have methods implemented (ShipOrder, RenameCustomer) to respond to domain events

Elements of Domain model: *Value objects*

- Objects of the model that typically have no concept of identity by reference
- Defined solely by it's property values, equality by value
- Typically implemented as immutable value objects to allow reuse/sharing in multiple entities
- Sales amount: amount and unit of measure (10 pounds)
- Currency amount: amount and currency unit (100,- €)
- Address
- May have methods, e.g. for changing unit of measure

Elements of Domain model: *Aggregates*

- Aggregates are combinations of Entities and value objects
  - Order, Order Lines and Customer
- From the outside represented as a single entity
- Supporting transactions on a set of Entities

Elements of Domain model: *Services*

- Functionality which implements relevant functionality of the domain model and conceptionally belongs to a number of objects / entities will be implemented as stand-alone services
- Services are typically state-less, reusable multiple times
- Methods reflect provided functionality
- Entities and value objects are passed in as parameters
- e.g. Price Calculation, Reservation of production capacities

Elements of Domain model: *Domain events*

- Events that domain experts care about
- Event represented by an object (event argument)
- Something happens in the domain which may cause multiple actions
  - Placing an order
  - Stock quantity changed
  - Renaming a customer
- Method of decoupling sub-systems, also with the goal to improve scalability
- When crossing system boundary, might require MQ

Elements of Domain model: *Modules*

- Modules separate the domain model into functional (not technical) parts
- Strong inner cohesion
- Loose coupling between modules
- Order entry module and customer maintenance module
- Modules support separation of developer team, reduce dependencies across the whole application
- Modules support easier analysis of impact of change

Agenda

- Introduction
- Domain-driven Design
- **OERA and CCS**
- Implementing modules
- Implementing domain events
- Implementing value objects
- Implementing Entities

OERA OpenEdge Reference Architecture

- Architecture blue print for service-oriented OpenEdge applications
- Initially released with OpenEdge 10.0 (15+ years)
- Primary goals at the time
  - AppServer enabling OpenEdge applications
  - Building non-monolithic OpenEdge applications
  - Supporting client flexibility
  - Providing guidance for use of the ProDataset
  - Providing guidance for use of OOABL (later, around OE10.1+)

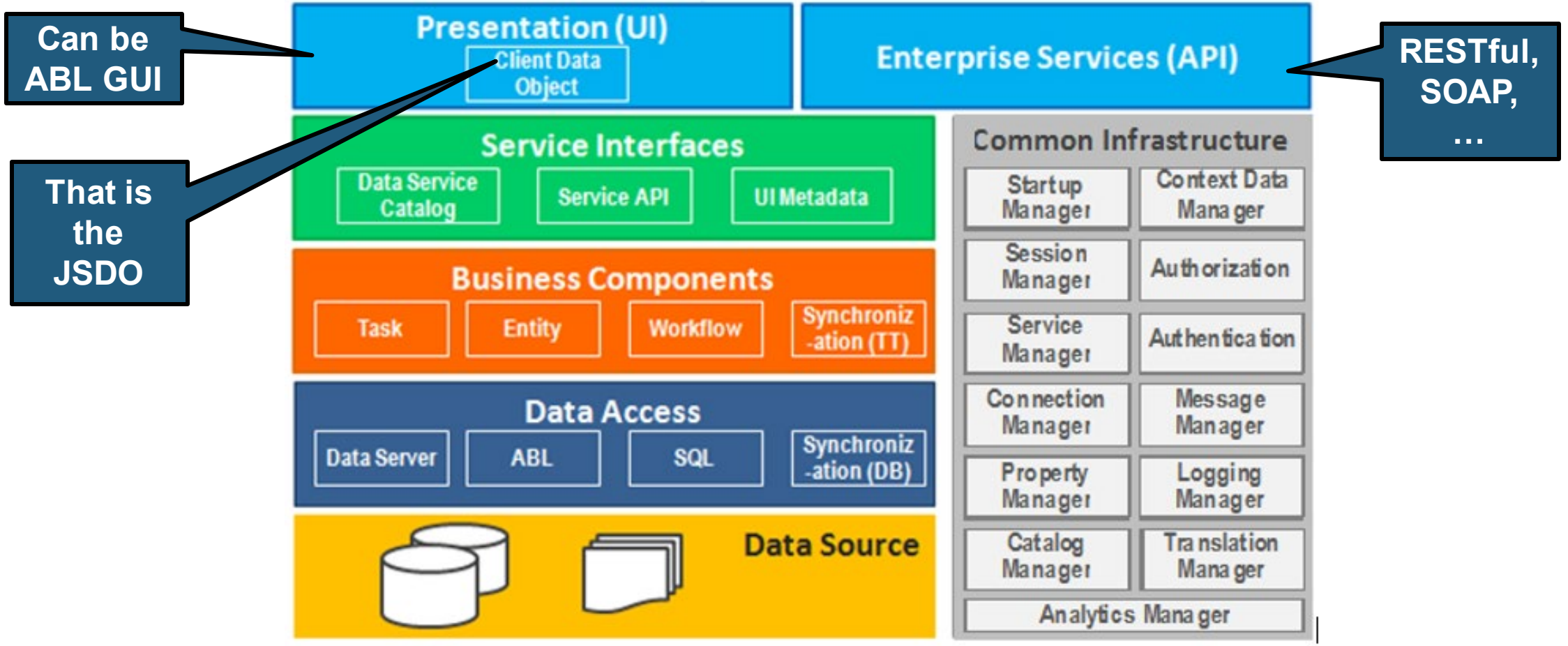
OERA today

- Fast forward to 2015 …
- Modernization of OpenEdge applications more relevant than ever; especially since Telerik acquisition and demands for UI flexibility
- OEAA – OpenEdge Application Architecture, redefining the OERA
- OERA back on focus, foundation of the **CCS (common component specification)** project as a vehicle for community and Progress driven architecture-spec efforts
- More detailed specs, rather than just programming samples
- Specs that an application or framework could be certified against
- CCS starting to influence *"in-the-box"* features

Business Entities

- Business Logic Component in the Business Service Layer
- Manages a set of database tables
  - Customer
  - Order/OrderLine/Item (read-only)
- CRUD actions (create, read, update, delete)
- Custom actions, verbs of the entity (PutCustomerOnCreditHold)
- Primary backend component for the JSDO
  - Kendo UI, Kendo UI Builder
  - NativeScript

The OpenEdge Application Architecture (OEAA)



CCS and DDD

- CCS does not define an implementation pattern for Domain-driven design per se
- CCS provides key building blocks for DDD implementation
  - Services and Service manager
  - Business Entities for Data Access and validation
  - Further infrastructure components which are required for almost every implementation – but irrelevant for the domain model. As they are not relevant to the domain experts
    - Context
    - Authentication, Authorization

Business Entities vs. Entities

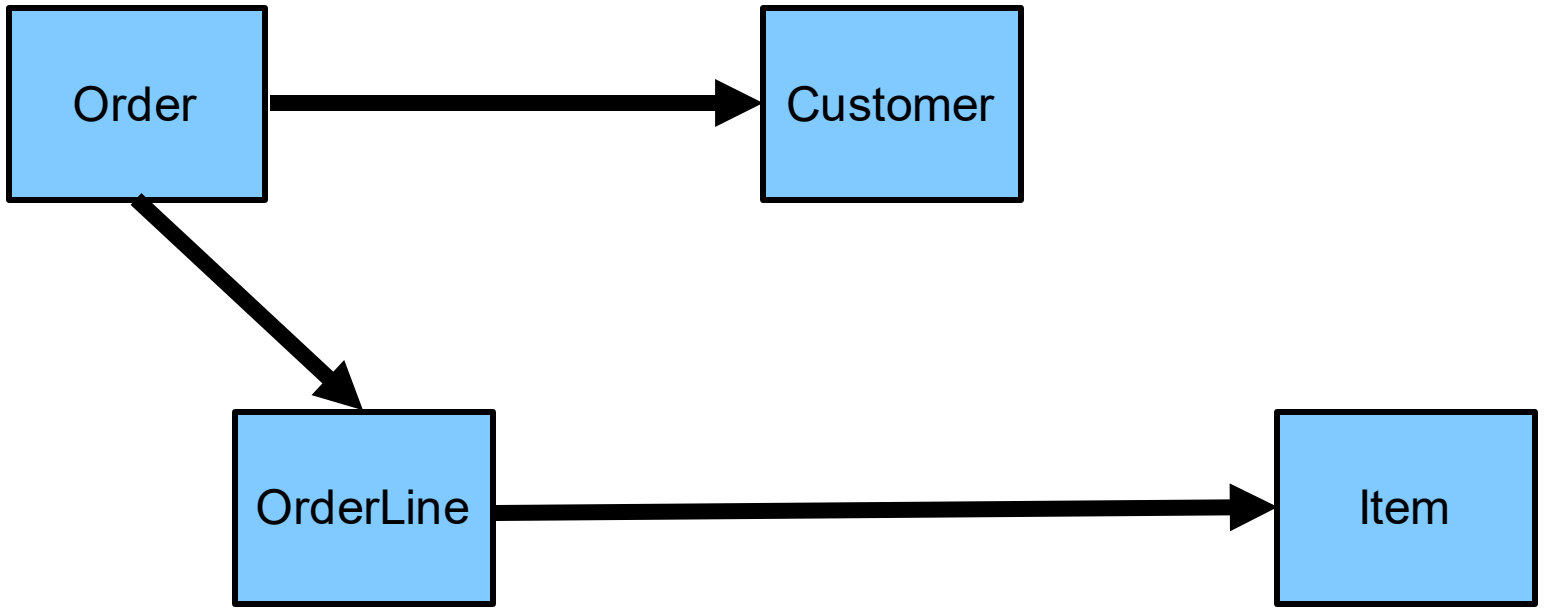
- Business Entities are not the same as Entities
- Business Entities are stateless service objects (CCS)
- Business Entities implement data retrieval logic (e.g. calculated fields, query optimization) and logic for storing records (e.g. validation)
- Domain-driven design relies on Data Access as well, however it's not the main focus, as it's not relevant for communication with domain experts
- Business Entities may be used for Data Access of Entities
  - in DDD repositories are used to retrieve and store Entities

Business Entities vs. Entities

- Business entities may be starting point for transforming ERD model into Domain model
- Single Database table as multiple temp-tables
- Multiple Database tables as single temp-table
- Reversing parent/child relations
  - In DB Order may be child of Customer (customer's order)
  - In Business Entity Customer may be child or Order (order's customer)
- Business Entities may read required data to assemble value objects in Entity
  - e.g. Quantity amount and unit of measure into additional temp-table fields

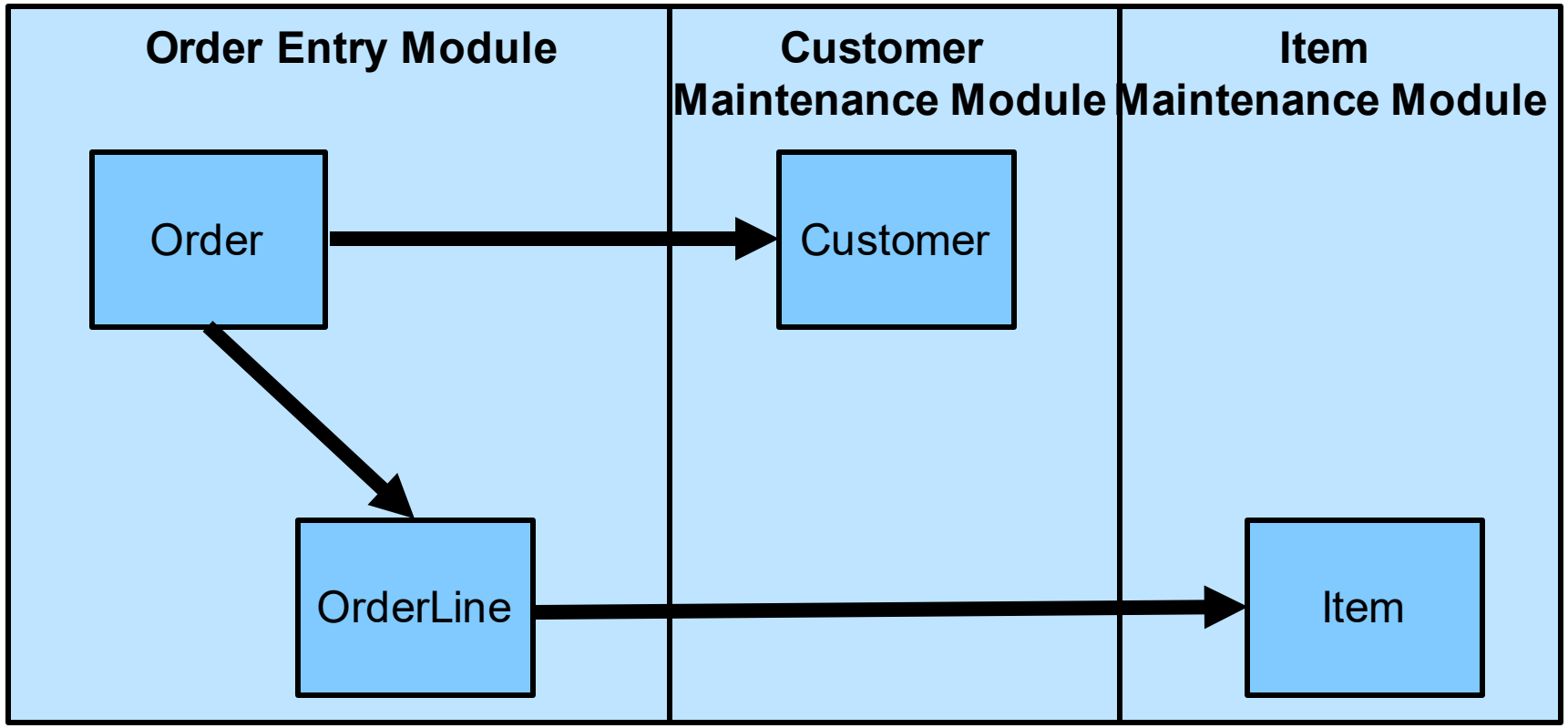
Practicability of module cohesion and loose coupling

- Business Entities may read data from multiple database tables



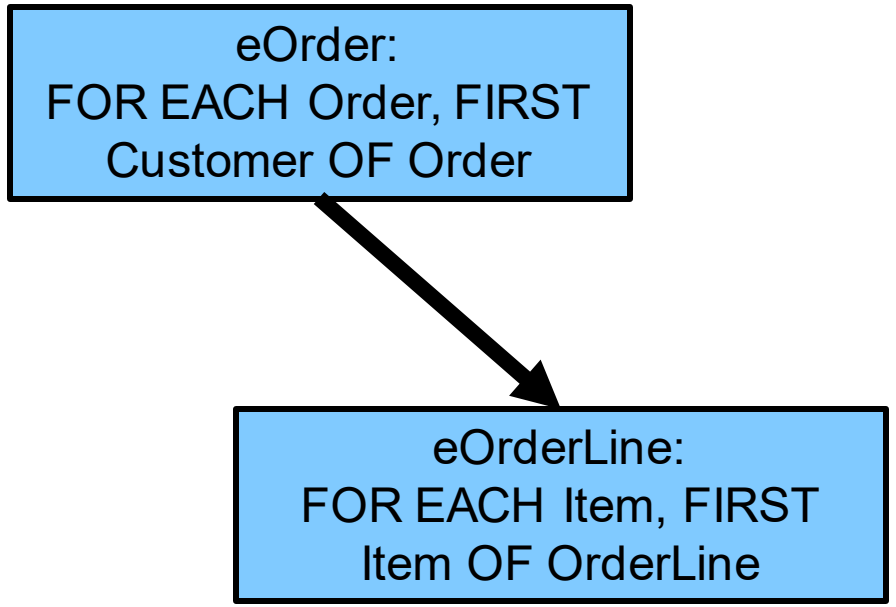
Practicability of module cohesion and loose coupling

- Those database tables may conceptually fit into different modules



Practicability of module cohesion and loose coupling

- Data Access through joined queries as DATA-SOURCE for ProDatasets
- Acceptable for read-only access to data from other modules
- It's unlikely to implement micro-services for each module. In ABL only a subset of database tables accessible



Practicability of module cohesion and loose coupling

- Loose coupling between modules would suggest retrieving data from different modules through service calls, e.g. Fetch Order and OrderLine first
  - Retrieve customers and items from services by set of required Id's
  - Either implemented as part of Business Entities or Repositories (DDD)
- Experience has shown however, that performance is better by an order of magnitude to resolve this as part of data access queries, crossing module boundaries

CCS Services and the Service Container

- Domain services are part of domain model
- Domain services are state-less classes implementing domain functions
- Good fit with CCS Services, and Service Manager

Service Manager

- Service Manager provides access to Services (that are not Managers)
- Factory for business services
- Calls their initialize() method
- Controls their life time
- Services typically launched at first request
- Services may be stopped (at the end of a request, after 1 hour, …)

Ccs.Common.IServiceManager

```
using CCS.Common.*.

interface CCS.Common.IServiceManager inherits IManager:

    method public IService getService( input poServiceClass as Progress.Lang.Class ).
    method public IService getService( input poServiceClass as Progress.Lang.Class,
                                       input pcInstanceName as character ).

    method public void stopService( input poServiceClass as Progress.Lang.Class,
                                    input pcInstanceName as character ).

end interface.
```

Obtaining reference to services

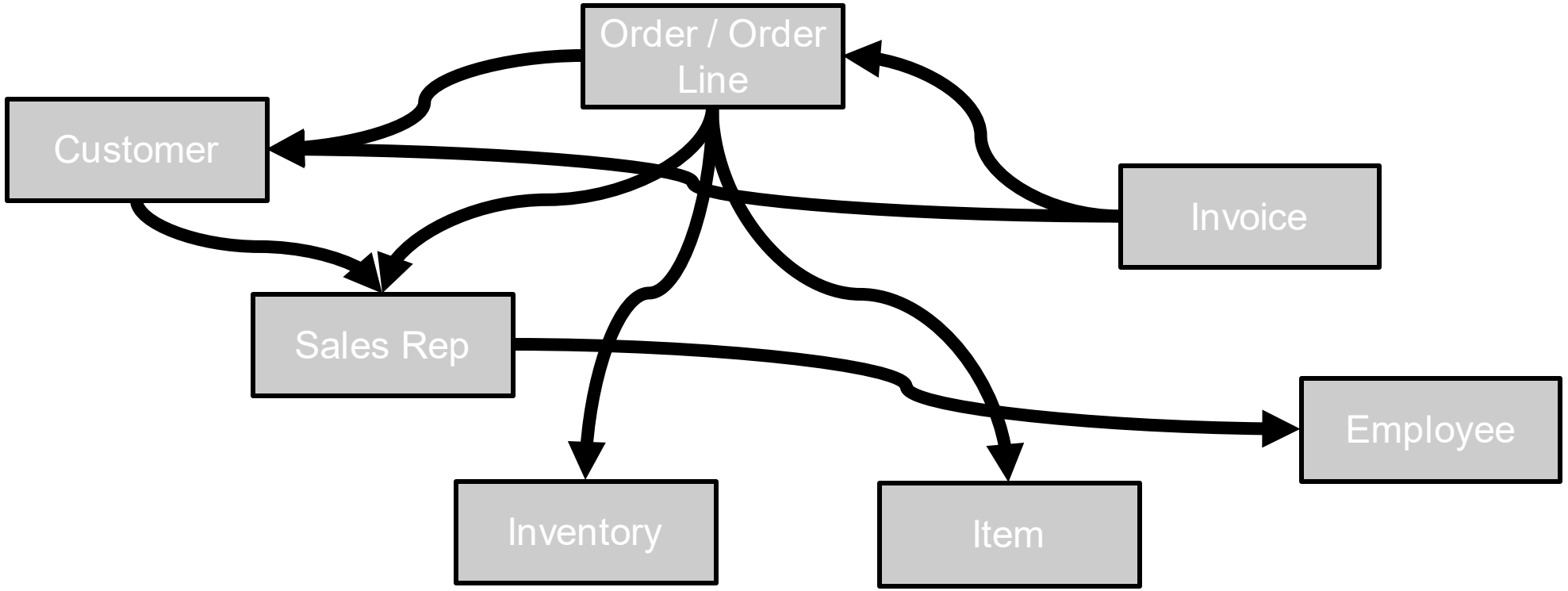
```
DEFINE VARIABLE oService AS IOrderTotalCalculatedFieldService NO-UNDO .

oService = CAST (Ccs.Common.Application:ServiceManager:getService
                                (GET-CLASS (IOrderTotalCalculatedFieldService)),
                 IOrderTotalCalculatedFieldService) .
```

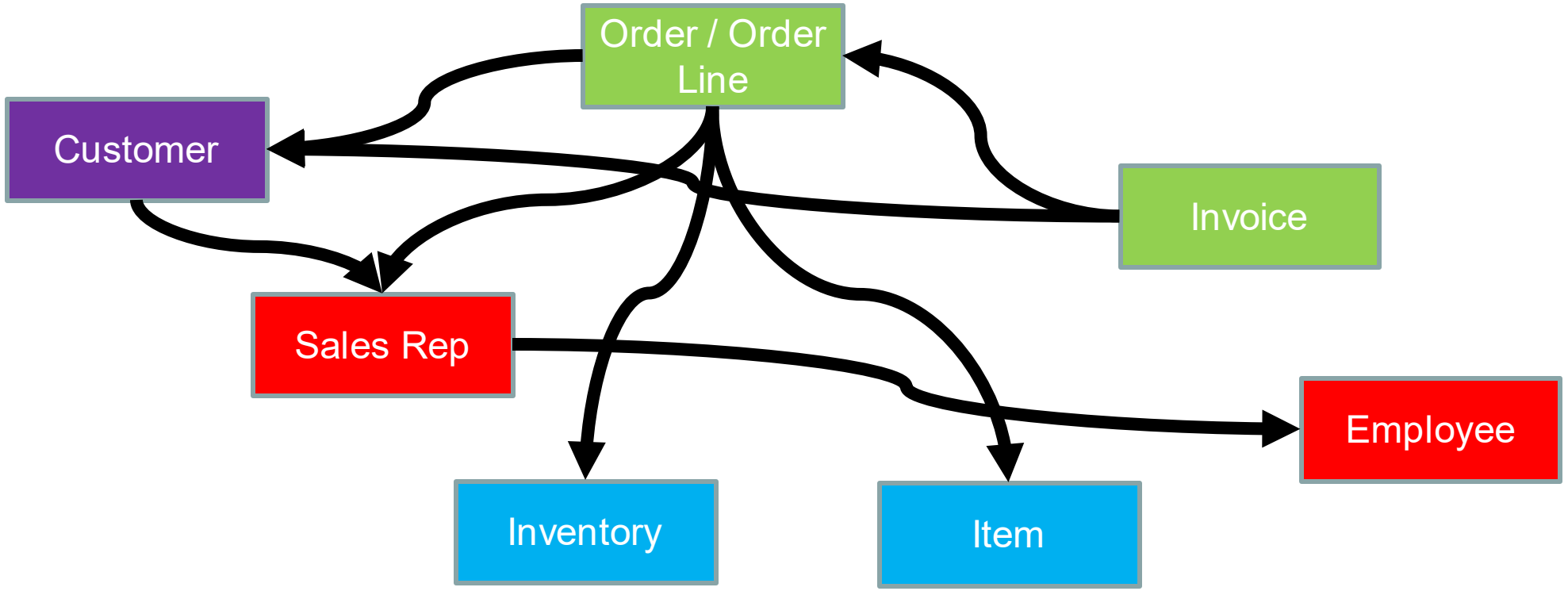
Agenda

- Introduction
- Domain-driven Design
- OERA and CCS
- **Implementing modules**
- Implementing domain events
- Implementing value objects
- Implementing Entities

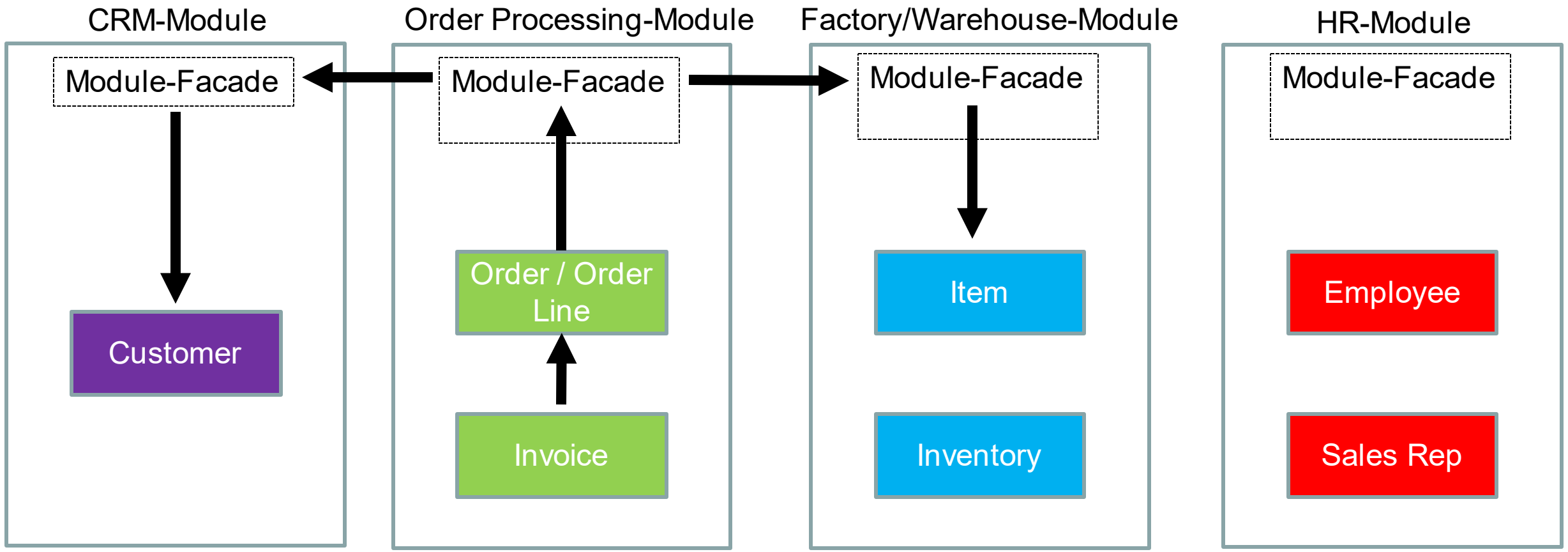
The OO Spaghetti Monster



Start ordering your objects in packages / modules



Divide and conquer – Modules as sub-systems



Facades

> The **facade pattern** (also spelled *façade*) is a software-design pattern commonly used with object-oriented programming. Analogous to a facade in architecture, a facade is an object that serves as a front-facing interface masking more complex underlying or structural code.
>
> A facade can improve the readability and usability of a software library by masking interaction with more complex components behind a single API
>
> https://en.wikipedia.org/wiki/Facade_pattern

Implementing modules

- Modules should define interaction patterns from the outside to the inside and within the module
- Outside should communicate through façade only
- Inside the module objects should be allowed to call directly into each other
- Domain-driven design accepts different implementation standards across modules, multiple agile teams agreeing on their own standards
- Modules can support modernization of application
  - Legacy functionality can be hidden behind facades
  - No need to modernize the whole application at once

Module facades as service

- Module facades should be implemented as services
- They represent a module to the outside
- Module facades are typically loaded when the application starts
- Module facades can subscribe to domain events (later)

Open/Close Principle

- Design as open for enhancement – while closed for modifications
- Design to a contract (Interface) on the sub-system level, not just a single class
- Modules manage complexity and impact of change
- A change in functionality in one module does not require changes to other modules
- Simplifies testing. Allows mocking of a whole sub-system

Open/Close Principle

- Business Requirement: Confirming an order needs to commit inventory/stock allocation
- If Order Business Entity would directly call into the Inventory/Stock Business Entity this would create a direct dependency between the two Business Entities
  - A change in the implementation of the Inventory/Stock Business Entity might affect the Order Business Entity
- If the Order Business Entity however, would publish a message using a Message Publisher infrastructure, the Inventory/Stock Domain may – or may not at it's own responsibility perform required action

Agenda

- Introduction
- Domain-driven Design
- OERA and CCS
- Implementing modules
- **Implementing domain events**
- Implementing value objects
- Implementing Entities

Implementing domain events

- Domain events are used to signal that something has happened (in the whole domain) that may be relevant for functionality within one or multiple modules
- Event itself more relevant than where it has happened
- Event may be raised due to action within an module
- Event may be raised due to action from outside
- Events primarily represented by message / payload

PUBLISH/SUBSCRIBE vs. Message Service

- We prefer to implement domain events through a Message Service (MessagePublisher)
- Listeners subscribe to message types (class, **interface**, OO type-compatibility)
- Publishers send message object (value object, PABLO) via Publish method of MessagePublisher service
- Single point of subscription for module façade
- Messages typically not based on Entities or Value objects as this might cause undesired dependencies

```
INTERFACE Consultingwerk.Framework.IMessagePublisher:

    /*------------------------------------------------------------------------
        Purpose: Publishes a Message to all subscriber that are subscribed to messag
                 of that type
        Notes:   Subscription is for the actual message type and child classes
        @param poMessage The Message to publish
    ------------------------------------------------------------------------*/
    METHOD PUBLIC VOID Publish (poMessage AS Progress.Lang.Object).

    /*------------------------------------------------------------------------
        Purpose: Subscribes an Message handler call back to the given Message type
        Notes:   Subscription is for the actual message type and child classes
        @param poCallback The reference to the callback instance
        @param poType The Message type to subscribe to
    ------------------------------------------------------------------------*/
    METHOD PUBLIC VOID Subscribe (poCallback AS IMessageSubscriber,
                                  poType AS Progress.Lang.Class).

    /*------------------------------------------------------------------------
        Purpose: Unsubscribes an Message handler call back to the given Message type
        Notes:
        @param poCallback The reference to the callback instance
        @param poType The Message type to unsubscribe from
    ------------------------------------------------------------------------*/
    METHOD PUBLIC VOID Unsubscribe (poCallback AS IMessageSubscriber,
                                    poType AS Progress.Lang.Class) .
```

PUBLISH/SUBSCRIBE vs. Message Service

- At system boundary messages may be sent to other systems
  - Via MQ
  - Via AppServer call from Service Adapter (representing remote services)

Demo

- Using the MessagePublisher

Agenda

- Introduction
- Domain-driven Design
- OERA and CCS
- Implementing modules
- Implementing domain events
- **Implementing value objects**
- Implementing Entities

Implementing value object

- Value objects are typically PABLO's (plain ABL objects), similar to POJO's (plain old Java object) or POCO's (plain old CLR object)
- Objects that are implemented to mainly store property values
- Equality based on values
- Consider overriding the Equals() method
- May contain methods, typically for basic calculations
  - Change unit of measure, change currency
  - Multiply, Add, …
  - Methods are not supposed to change properties of value object, rather return new object instance

```
CLASS Consultingwerk.SmartComponentsDemo.DDD.Price:

    DEFINE PUBLIC PROPERTY Amount AS DECIMAL NO-UNDO
    GET.
    PRIVATE SET.

    DEFINE PUBLIC PROPERTY CurrencySymbol AS CHARACTER NO-UNDO
    GET.
    PRIVATE SET.

    CONSTRUCTOR PUBLIC Price (pdeAmount AS DECIMAL,
                              pcCurrencySymbol AS CHARACTER):

        ASSIGN THIS-OBJECT:Amount         = pdeAmount
               THIS-OBJECT:CurrencySymbol = pcCurrencySymbol .

    END CONSTRUCTOR.
```

```
METHOD PUBLIC Price CurrencyConversion (pcNewCurrencySymbol AS CHARACTER):

    DEFINE VARIABLE oRateService AS ICurrencyRateService NO-UNDO .

    oRateService = {Consultingwerk/get-service.i ICurrencyRateService} .

    RETURN NEW Price (THIS-OBJECT:Amount *
                      oRateService:GetConversionRate (THIS-OBJECT:CurrencySymbol,
                                                      pcNewCurrencySymbol),
                      pcNewCurrencySymbol) .
END METHOD.
```

- IF oPrice1:Equals (oPrice2) THEN …

```
METHOD PUBLIC OVERRIDE LOGICAL Equals (oObject AS Progress.Lang.Object):

    DEFINE VARIABLE oPrice AS Price NO-UNDO .

    IF NOT VALID-OBJECT (oObject) THEN
        RETURN FALSE .

    IF NOT TYPE-OF (oObject, Price) THEN
        RETURN FALSE .

    oPrice = CAST (oObject, Price) .

    RETURN      THIS-OBJECT:Amount         = oPrice:Amount
           AND THIS-OBJECT:CurrencySymbol = oPrice:CurrencySymbol.

END METHOD.
```

Implement your own equality rules, eventually changing currency

- MESSAGE STRING (oPrice)
- MESSAGE oPrice:ToString()
- MESSAGE oPrice

```
METHOD PUBLIC OVERRIDE CHARACTER ToString():

    RETURN SUBSTITUTE ("&1 &2":U,
                       THIS-OBJECT:Amount,
                       THIS-OBJECT:CurrencySymbol).
END METHOD.
```

Why immutable?

- Because the value 42 cannot be changed as well
- A value object in combination with the properties represents a single value
- Value objects may be reused
  - May improve performance
- When reusing value objects, a change to a property would affect all references to the single value object

Why immutable?

```
DEFINE VARIABLE oOrderLine1 AS OrderLine NO-UNDO .
DEFINE VARIABLE oOrderLine2 AS OrderLine NO-UNDO .

oOrderLine1 = NEW OrderLine () .
// ...
oOrderLine1:Price = NEW Price (100, "USD":U) .


oOrderLine2 = NEW OrderLine () .
// ...
oOrderLine2:Price = oOrderLine1:Price .

// this would change OrderLine2:Price:Amount as well
oOrderLine1:Price:Amount = 42 .
```

Implementing Equals() via ABL Reflection

```
METHOD PUBLIC OVERRIDE LOGICAL Equals (poObject AS Progress.Lang.Object):
    DEFINE VARIABLE oProperties AS Progress.Reflect.Property EXTENT NO-UNDO .
    DEFINE VARIABLE i           AS INTEGER                    NO-UNDO .

    IF NOT VALID-OBJECT (poObject) THEN
        RETURN FALSE .

    IF poObject = THIS-OBJECT THEN
        RETURN TRUE .

    IF NOT poObject:GetClass():IsA (THIS-OBJECT:GetClass()) THEN
        RETURN FALSE .

    oProperties = THIS-OBJECT:GetClass ():GetProperties (Progress.Reflect.Flags:Public OR Progress.Reflect.Flags:Instance

    propertyLoop:
    DO i = 1 TO EXTENT (oProperties):

        IF NOT oProperties[i]:CanRead THEN NEXT propertyLoop .

        IF oProperties[i]:DeclaringClass = GET-CLASS (Progress.Lang.Object) THEN NEXT propertyLoop .

        IF oProperties[i]:Get (THIS-OBJECT) <> oProperties[i]:GET (poObject) THEN RETURN FALSE .
    END.

    RETURN TRUE.

END METHOD.
```

Agenda

- Introduction
- Domain-driven Design
- OERA and CCS
- Implementing modules
- Implementing domain events
- Implementing value objects
- **Implementing Entities**

Implementing Entities

- Entities represent the core implementation blocks of the Domain model
- Entities typically represent data from the database
  - Yes – domain experts might not care
  - But they do care that today's order is still available tomorrow
- Equality is defined based on primary unique key values or similar
- Depending on requirements for abstraction, Entities can be built on top of a Business Entity ProDataset schema
- Entities might implement further domain logic
- Use repositories to retrieve and save Entities to the database

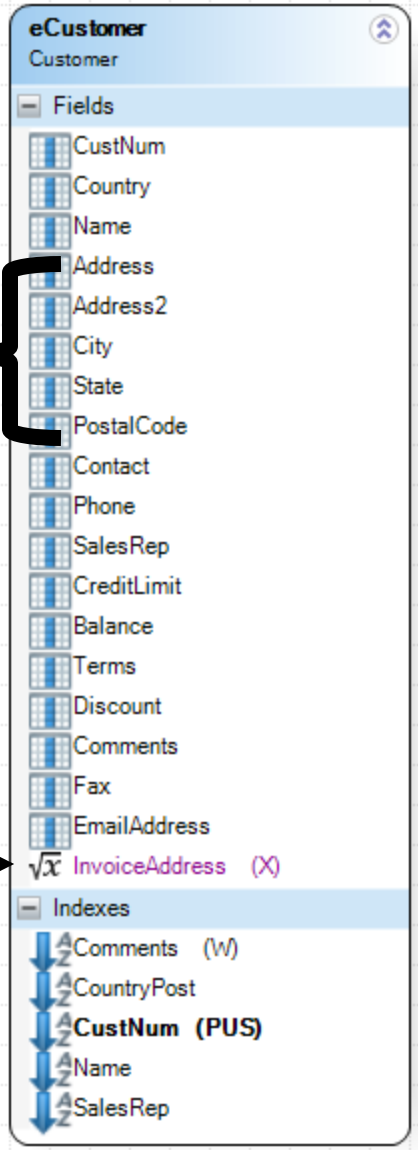
Entity based on ProDataset Temp-Table

```
oCustomer:
```

- Available : logical - TableModel
- Balance : decimal - CustomerTableModel_Generated
- Batching : logical - TableModel
- BufferError : logical - TableModel
- BufferErrorString : character - TableModel
- BufferHandle : handle - TableModel
- BufferModelGcMode : BufferModelGcModeEnum - TableModel
- BufferName : character - TableModel
- BufferRejected : logical - TableModel
- Comments : character - CustomerTableModel_Generated
- Contact : character - CustomerTableModel_Generated
- CreditLimit : decimal - CustomerTableModel_Generated
- CustNum : integer - CustomerTableModel_Generated
- DatasetModelMode : DatasetModelModeEnum - TableModel
- DatasetModelPerformer : DatasetModelPerformer - TableModel
- Discount : integer - CustomerTableModel_Generated
- EmailAddress : character - CustomerTableModel_Generated
- Fax : character - CustomerTableModel_Generated
- FillChildTables : character - TableModel
- Filter : CustomerTableModelFilter - CustomerTableModel_Generated
- InvoiceAddress : Consultingwerk.SmartComponentsDemo.DDD.Customer.Address - CustomerTableModel_Generated
- ModelType : TableM…elTypeEnum - TableModel
- Name : character - CustomerTableModel_Generated
- Next-Sibling : Progress.Lang.Object - Progress.Lang.Object
- Phone : character - CustomerTableModel_Generated

Database fields „hidden“ from Entity as they are represented by Value Object instead

Value Object Column, not present in database →



Demo

- Review Customer Entity Design based on ProDataset
- Review Address value object mapping
- Review Address

Questions

- Email: info@consultingwerk.com
- www.consultingwerk.com
- https://www.youtube.com/consultingwerk