DOTNEXT

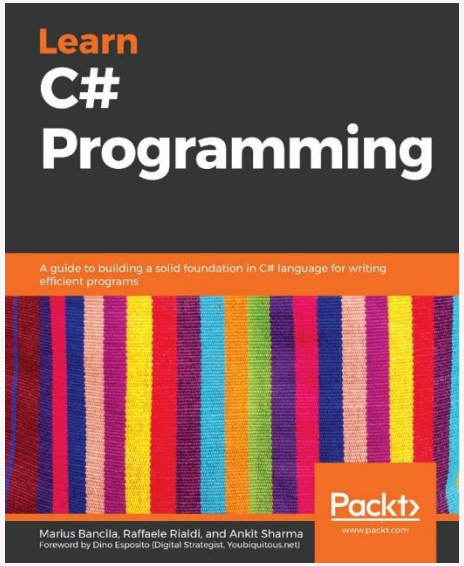
Overcome model versioning nightmare using Semantic Driven Modeling (SDM) in distributed systems

Raffaele Rialdi - Senior Software Architect

@raffaeler
raffaeler@vevy.com

Who am I?

- Raffaele Rialdi, Senior Software Architect in Vevy Europe – Italy
  - @raffaeler also known as "Raf"
- Consultant in many industries
  - Manufacturing, racing, healthcare, financial, …
- Speaker and Trainer around the globe
  - Italy, Romania, Bulgaria, Russia, USA, …
- Proud member of the great Microsoft MVP family since 2003



Agenda

- Ultimate goal: resolve versioning in REST based services
- We start looking at the scenarios that SDM can resolve
- How to build the mappings between two models
- Digging into the Semantic Driven Modeling [SDM] machinery
- Demo!

Main scenarios relevant to SDM

How can we map these two models?

```
public class Order {
  Guid Id
  string Reference
  List<OrderItem> OrderItems
}
```

```
class OrderItem {
  Guid Id
  Article Article
  Company Customer
  decimal Quantity
  decimal Price
  decimal Discount
}
```

```
class Article {
  Guid Id
  string Name
  string Description
  DateTime ExpirationDate
}
```

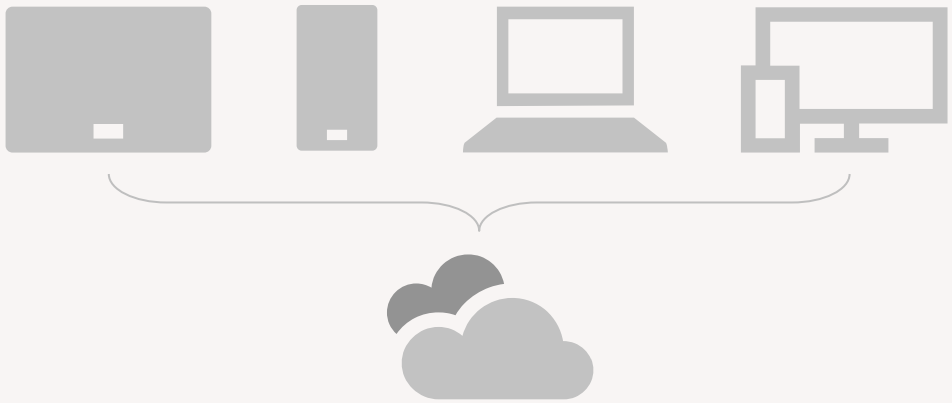
```
class Company {
  Guid Id
  string Name
  string Description
  Address Address
}
```

```
class Address {
  Guid Id
  string Street
  string City
  string State
  string Country
}
```

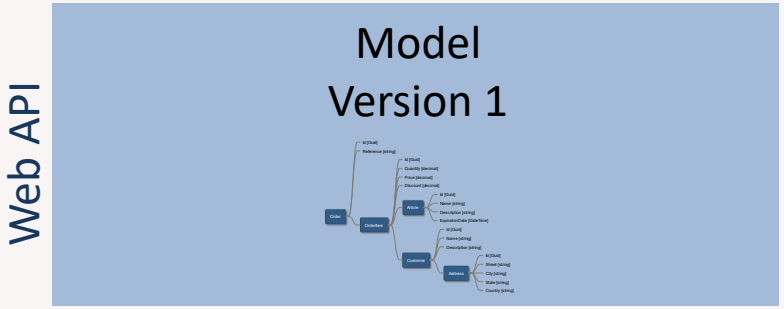
```
class OnlineOrder {
  Guid Id
  string Description
  List<OrderLine> OrderLines
}
```

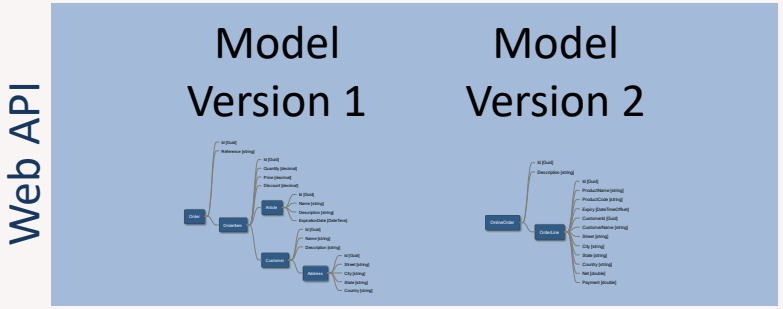
```
class OrderLine {
  Guid Id
  string ProductName
  string ProductCode
  DateTimeOffset Expiry
  Guid CustomerId
  string CustomerName
  string Street
  string City
  string State
  string Country
  double Net
  double Payment
}
```

Scenario 1: One service updates the public model



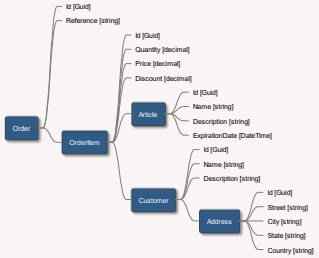
- New endpoints
- Maintaining old model
- Multiple controllers

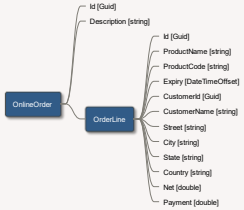




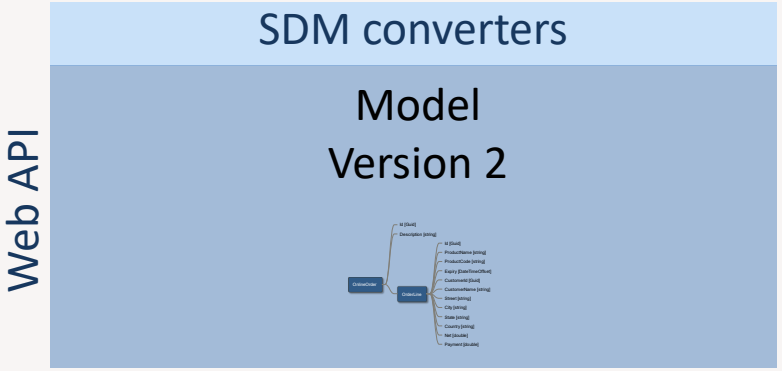
Scenario 1: The SDM "server-only" solution

JSON of Model Version 1



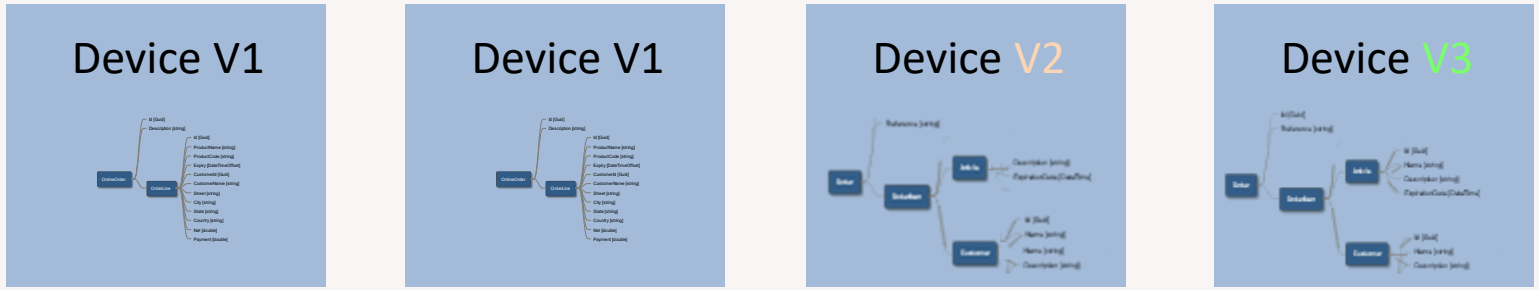


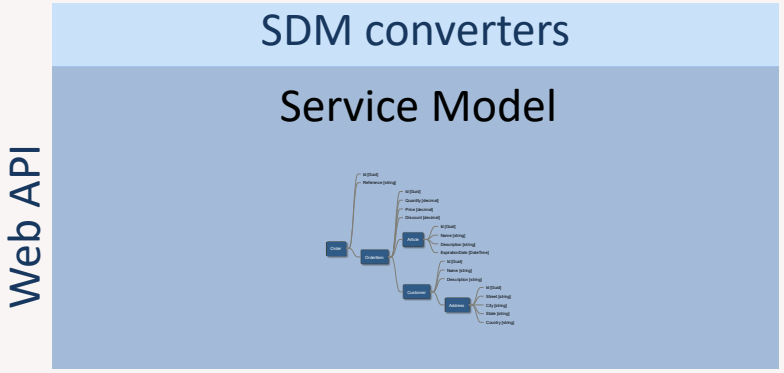
JSON of Model Version 2



- Conversion happens on the server
- Validate the conversions with tests
- Service code is much simpler
- No DTO, no manual mapping code

Scenario 2: IoT devices publishing data



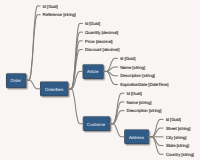


- Convert straight from JSON
- No DTO, no manual mapping code
- Mappings can be validated at dev time

Scenario 3: Event Sourcing / Bus / Queue

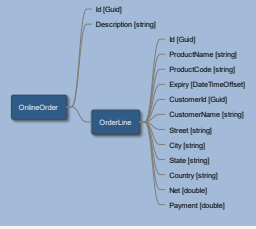
Write channel
(commands)

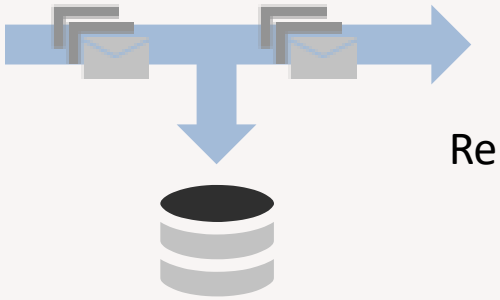
Multiple
Reading models



SDM converters

Service Model





Relational DB

NoSql Database
(Event / Messages slices)

SDM converters

Entity Framework
Model

- Ability to read historic slices

- Ability to expose multiple RM
- User configured reading models

Scenario 4: Microservice-based architecture

Service 1 Model

SDM

Service 2 Model

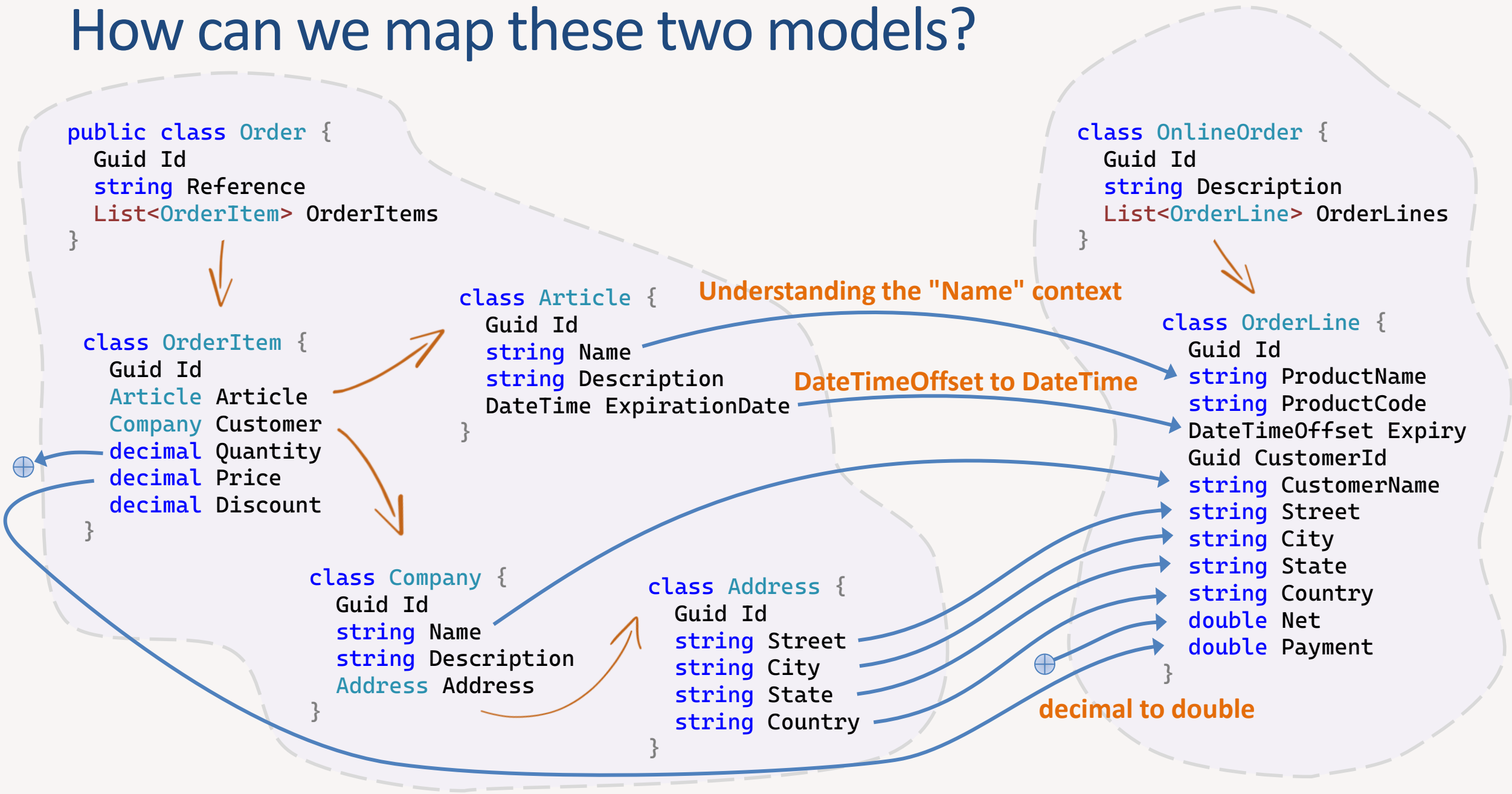
SDM

Service 3 Model

SDM

Service 4 Model

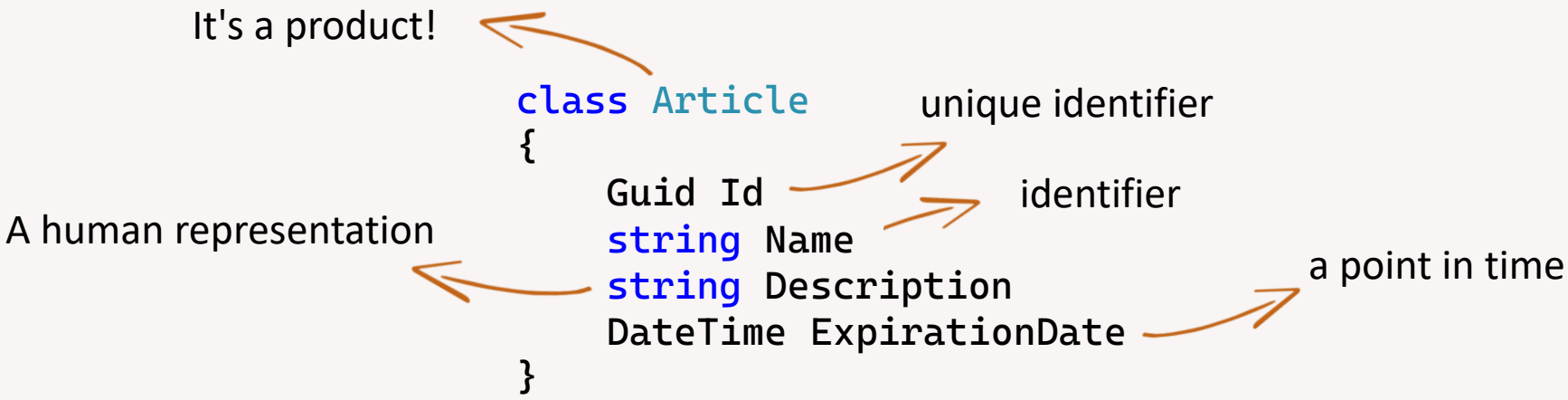
How can we map these two models?



Automatic building the mappings

Semantic metadata

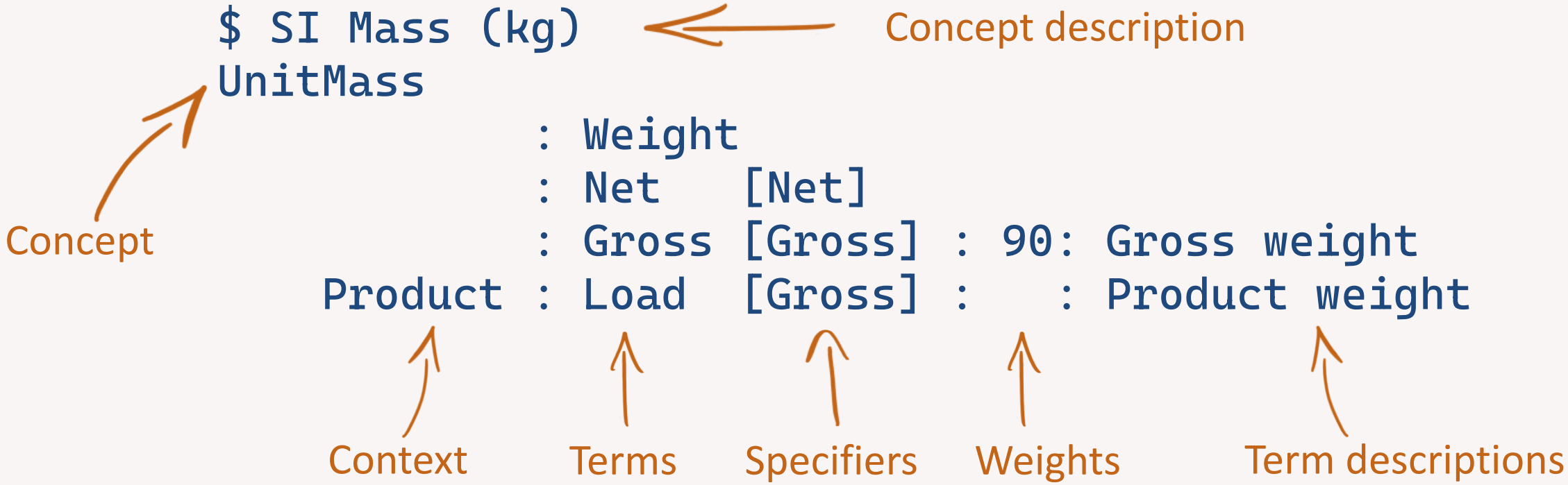
- We try to **understand** the **concepts** behind the property/class names



- How can a computer know that?
  - Using a vocabulary that is specific to our business domain
  - Preserve the **context** for every navigation or property
    - Name is not just a "Name" but the "Product Name"

The Domain Specific Language

- Write down your business glossary in a text file
- Group the words by meaningful concepts
- Let the C# Code Generator create the strongly-typed classes

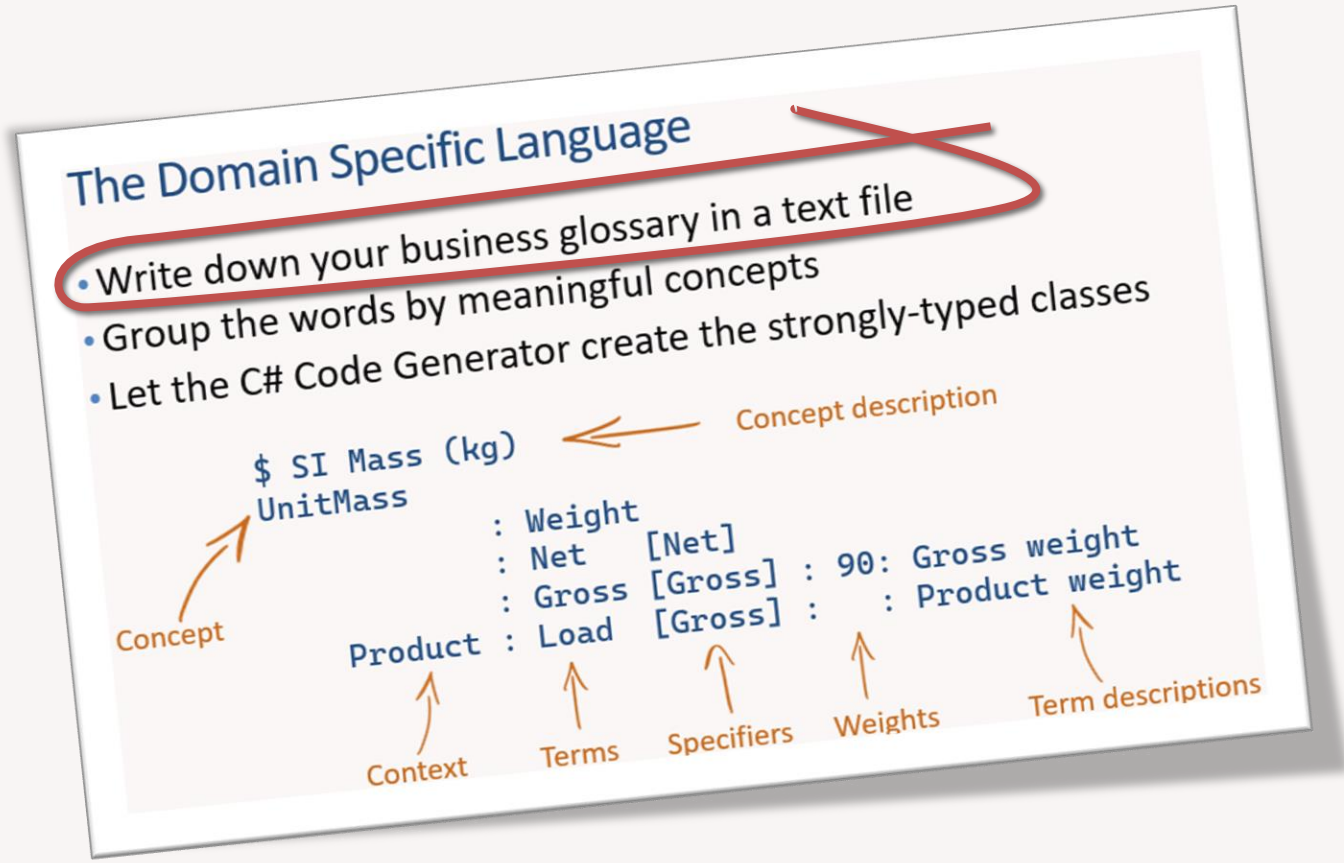


Ever heard about DDD?

- Doesn't this sound as the "Ubuiquitous Language"?

«Domain experts should object to terms or structures that are awkward or inadequate to convey domain understanding; developers should watch for ambiguity or inconsistency that will trip up design. With a ubiquitous language, the model is not just a design artifact. ... Play with the model as you talk about the system.»

Eric Evans,
Domain-Driven Design: Tackling Complexity in the Heart of Software, 2003



He writes detailed comments and uses descriptive variable names

The SDM Domain elements

- Domain links Terms, Concepts, Concept Specifiers and Weights

- This is the place where you may want to add:
  - Policies, Validations and Rules

- Examples:
  - Policy: our measurements are in SI and not Imperial
  - Validation: every weight property should use the decimal data type
  - Rule: only map an UniqueIdentifier with another UniqueIdentifier

ASP.NET Core specific code

- Two (input/output) formatters to negotiate the version
  - Using standard headers: Accept and Content-Type
  - The format is:

    `application` / `sdm . {domain name} . {TsId}` + `json`

    `type` `Subtype` `suffix`
- Using standard headers: Accept and Content-Type
- **OpenAPI**
  - SDM produces additional metadata for projected types
- An additional endpoint with SDM metadata may be very useful

What about GraphQL?

- Two options:
  - Use both: they play well together
  - Use SDM only on the server side: if you just need to reduce the data
    - Just remove one or more mappings

- Remember:
  - GraphQL does NOT change the graph shape

Benchmarks!

Serialization

code generated

	Method	Mean	Error	StdDev
.NET	PlainOrders	9.702 µs	0.1862 µs	0.1992 µs
.NET	PlainOnlineOrders	6.832 µs	0.1318 µs	0.1518 µs
SDM	SemanticOrderToOnlineOrder	6.108 µs	0.1186 µs	0.1318 µs
SDM	SemanticOnlineOrderToOrder	6.064 µs	0.0987 µs	0.0923 µs

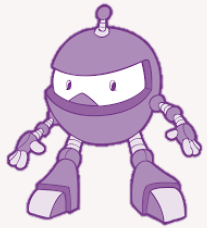
Deserialization

code generated

	Method	Mean	Error	StdDev
.NET	PlainOrders	18.16 µs	0.363 µs	0.496 µs
.NET	PlainOnlineOrders	10.23 µs	0.172 µs	0.205 µs
SDM	SemanticOrderToOnlineOrder	32.91 µs	0.638 µs	0.655 µs
SDM	SemanticOnlineOrderToOrder	28.59 µs	0.519 µs	0.433 µs

Without code-generation, the semantic version raises of 3 orders of magnitude (~ x1000)

Questions?



Thank you!

@raffaeler

raffaeler@vevy.com

Microsoft Most Valuable Professional