



Events at the tip of your fingers

Using CQRS and event sourcing
on a mobile environment



Who am I ?



Next → March 27-29 2015



Technical lead



Architecture gardener



Agile coach



cpontet@agilepartner.net



[@cpontet](https://twitter.com/cpontet)



[linkedin.com/pub/cédric-pontet](https://www.linkedin.com/pub/cédric-pontet)



[google.com/+CédricPontet](https://plus.google.com/+CédricPontet)

Who are you ?

Whose role is



Achitect



Mobile developer



iOS



Android



Windows Phone



Windows Mobile



.NET developer



Java developer

Who is familiar with



DDD



CQRS



Event sourcing

I feel for
these guys

What seems to be the problem ?

Implementing a mobile app to help distributing mail and parcels

What is the problem domain ?



Let's write some user stories...

DELIVERING MAIL

As a mail delivery agent

I want to track the mail that is going to be distributed during my daily delivery tour

So that customers know what is going on with their mail

DELIVERING PARCELS

As a parcel delivery agent

I want to be guided through my daily delivery truck tour

So that parcels get delivered as fast and efficiently as possible

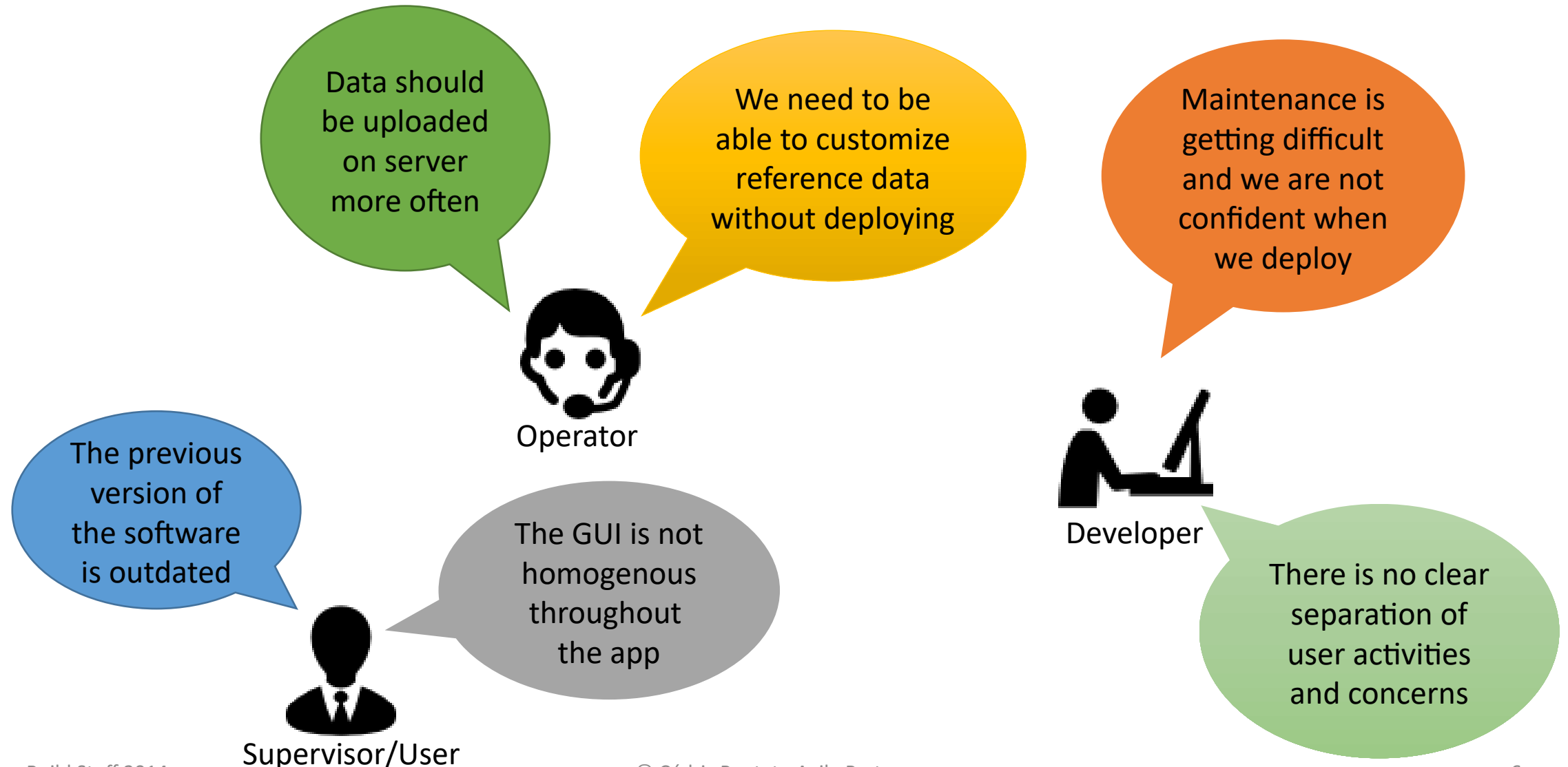
SUPERVISING PARCEL WAREHOUSE

As a parcel warehouse supervisor

I want to route the parcels to the right truck

So that they get delivered on time

Why rebuild an existing software ?

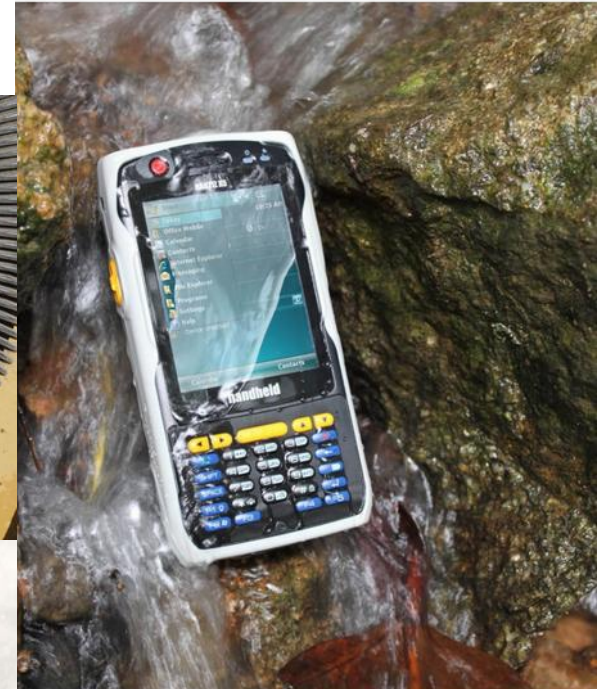


Meet the Nautiz X5

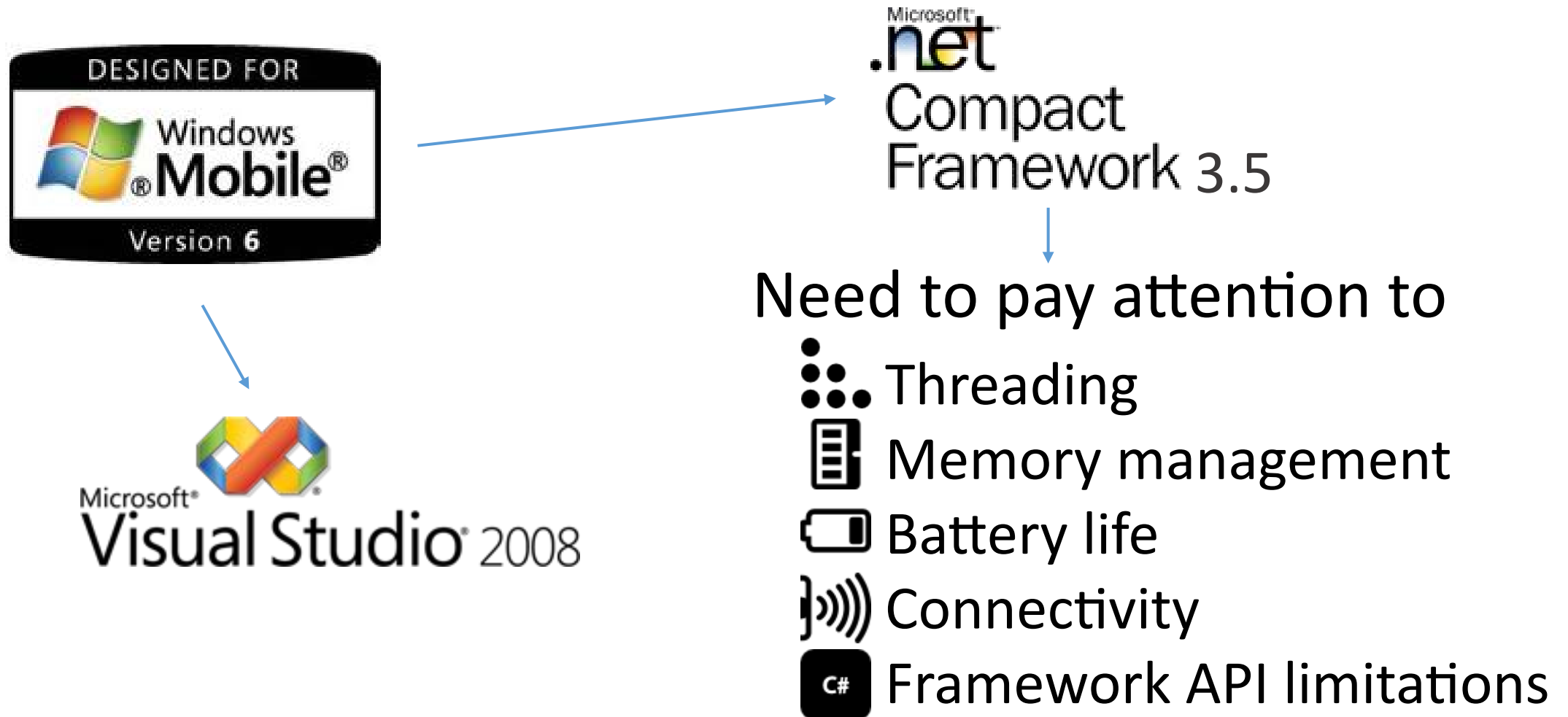


- Professional mobile device
 - 806 MHz CPU
 - 256 MB RAM
 - 480 x 640 touch-screen
 - WiFi / GPRS / Bluetooth / GPS
 - Infra-red scanner
- Would probably survive a plane crash
 - Or even drowning
 - Operating -20 °C to 55 °C

I am not kidding



But... what are the constraints ?



Meet my new friends

NEventStore



OpenNETCF Consulting

Managed Code in the Embedded World

resco.net
technology on the move

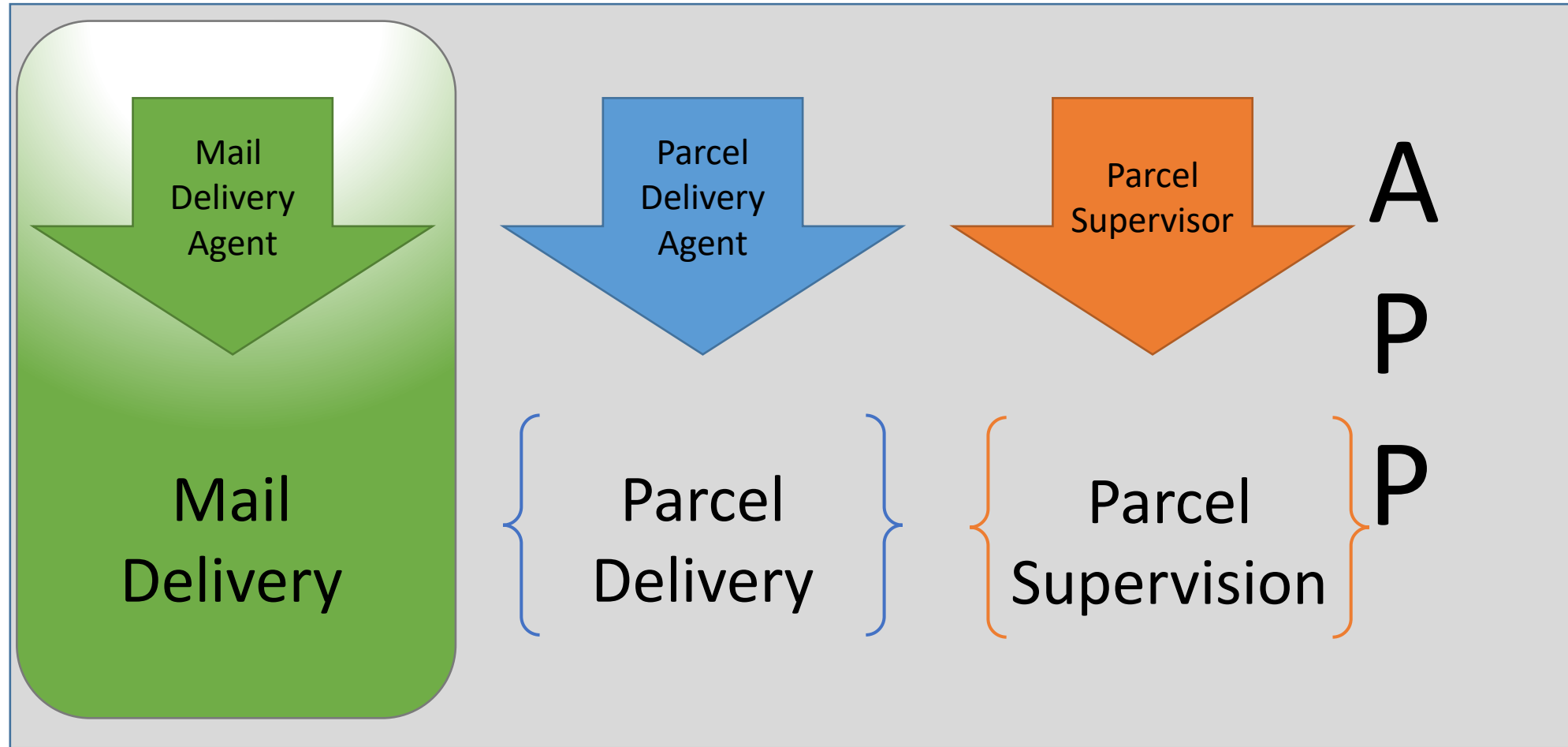


MobiControl[®]

Why events ?

Designing software with events

3 roles == 3 modules



Mail delivery



More user stories...

LOG ON THE APPLICATION

As a mail delivery agent

I want to scan the bar code uniquely identifying my daily tour

So that I can initiate the tour and start enlisting products to be delivered

PREPARE DAILY TOUR

As a mail delivery agent

I want to scan the bar codes of all the products that I have to distribute

So that I can enlist them as being part of my tour

SET MAIL STATUS

As a mail delivery agent

I want to scan the bar codes of the products to distribute to a given customer

So that I can indicate whether it has indeed been delivered or not

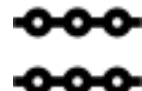
COLLECT MAIL BOX

As a mail delivery agent

I want to scan the bar code of a mail box

So that I can mark it as collected

Event oriented by essence

 Thinking with events thanks to Greg Young

 What can possibly happen to a piece of mail ?

 Mail is ***delivered*** to the recipient when at home

 The recipient is ***notified*** that he has mail awaiting

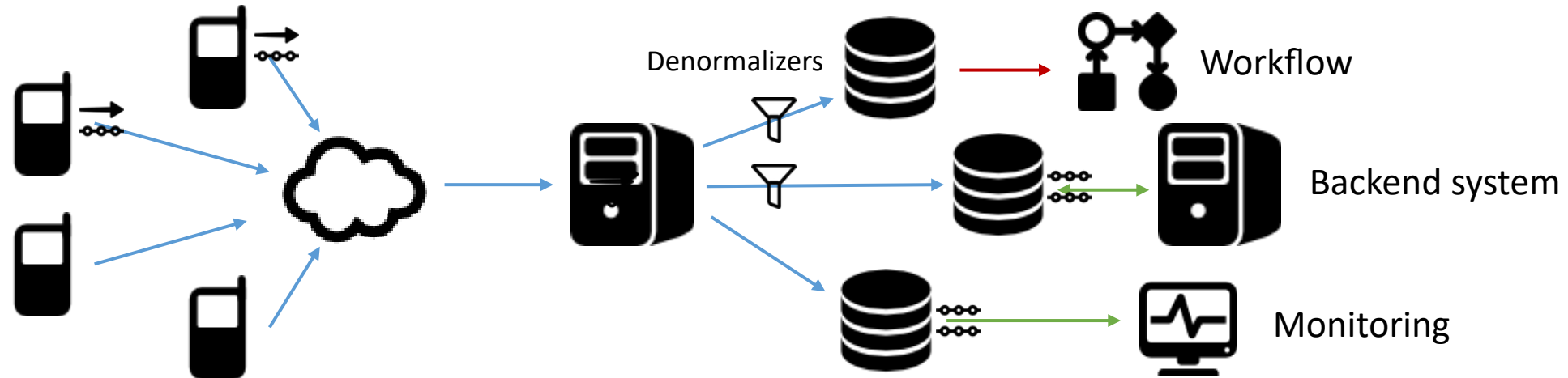
 Mail is ***returned to sender*** when the recipient is unknown

 Mail is ***sent back to distribution*** to be delivered later

 Mail is ***routed to a pickup station***

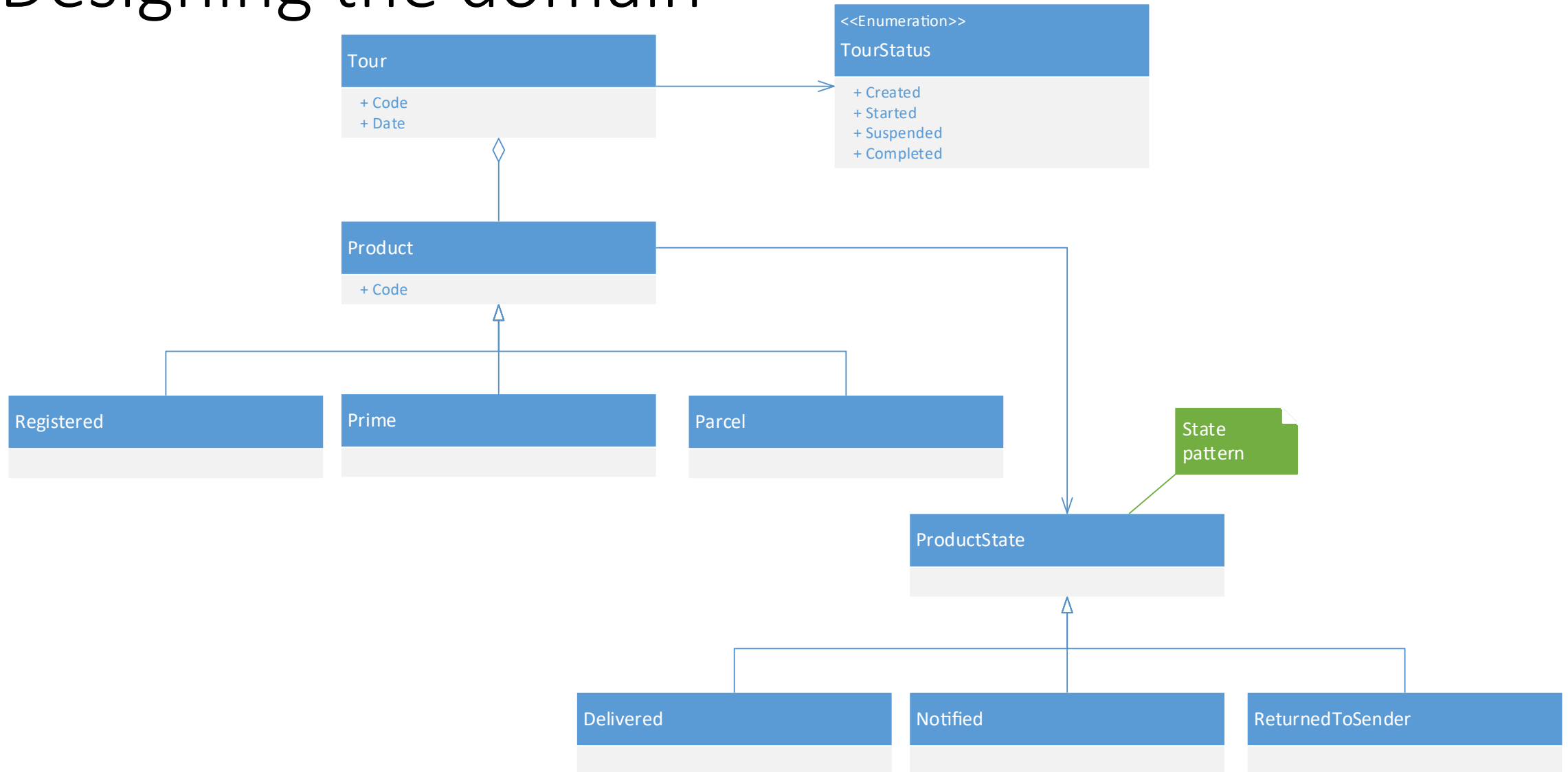
All past tense verbs

Eventual consistency is not an issue



Devices do not need to know what is going on in the rest of the world
All they need is to share the same business rules
Data has to be upload to the server eventually

Designing the domain



Designing the domain with events



Unfortunately did not know about Event Storming at that time

Heard about it at Build Stuff 2013

Thanks [@ziobrando](#)



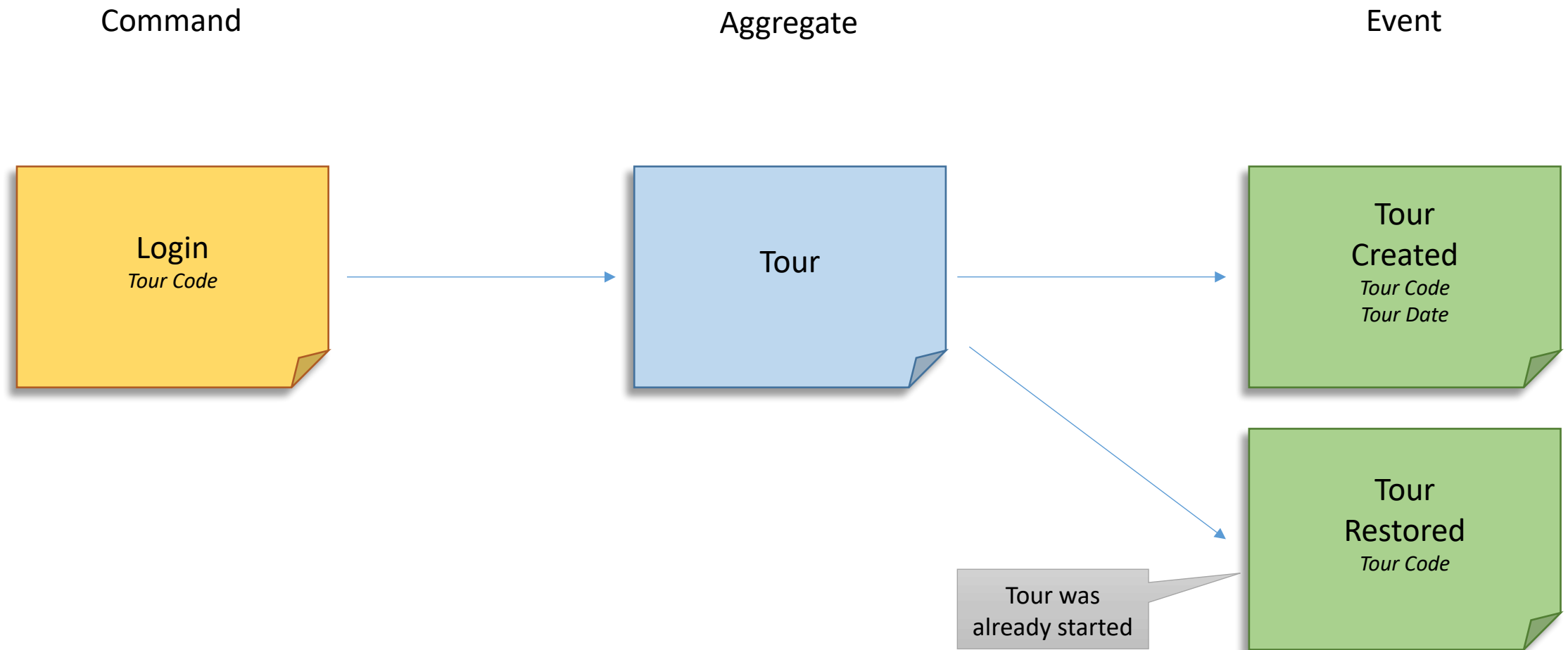
Just started thinking about

Domain events

Commands

Aggregates

Login

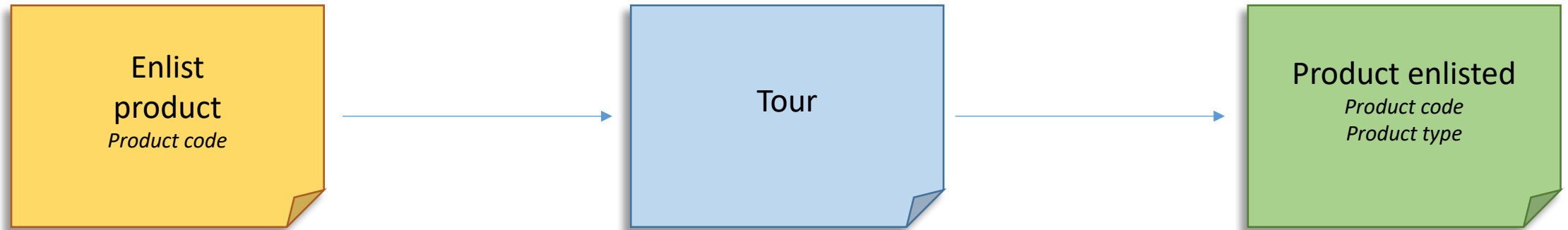


Enlist products

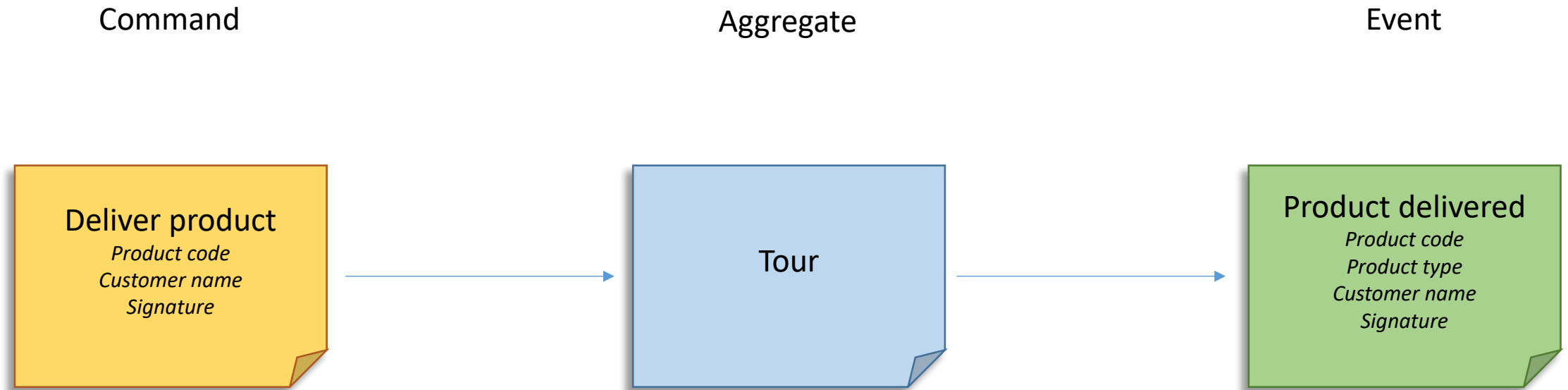
Command

Aggregate

Event



Deliver product to customer

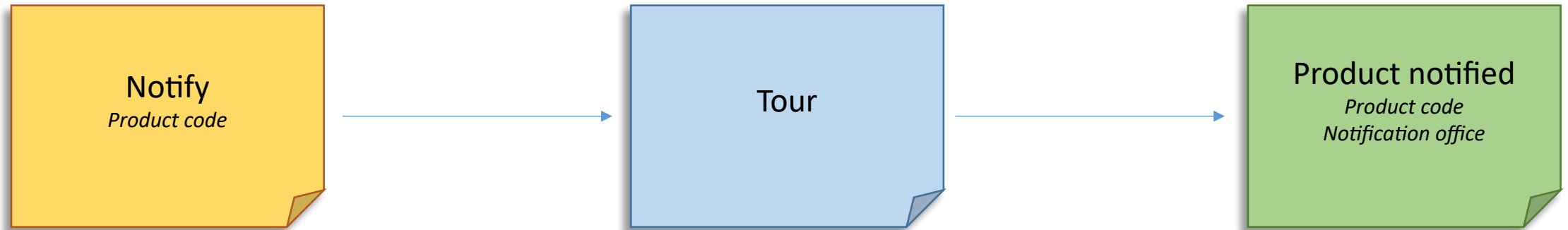


Notify

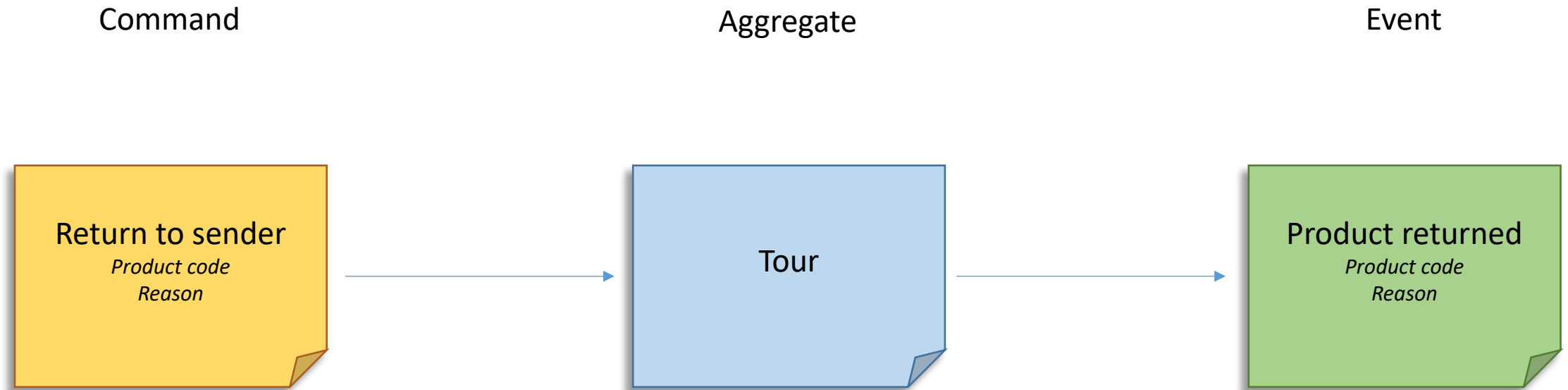
Command

Aggregate

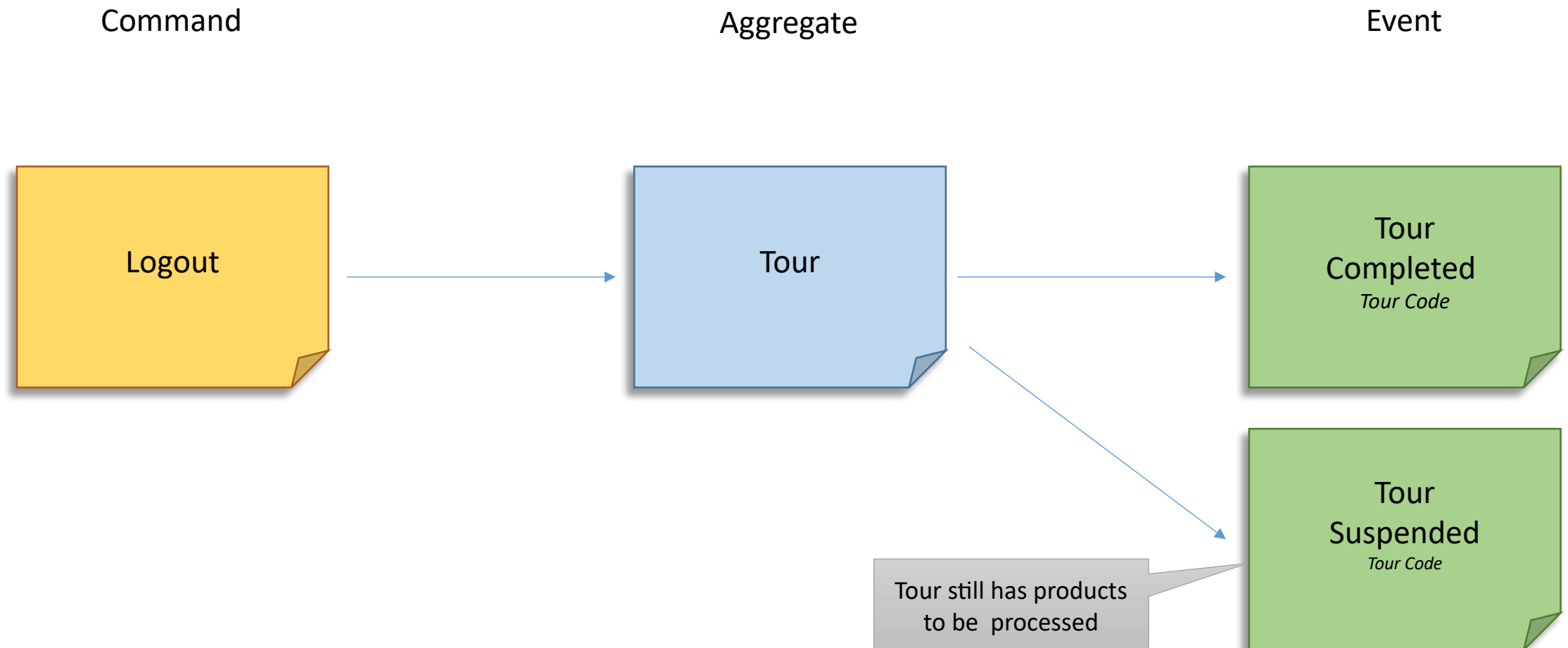
Event



Return product to sender



Logout



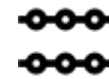
Domain rules : product type * delivery option

	Remis			Avisé	
	Destinataire	Boîte	Autre	Client absent	PackUp 24/24
Recommandé (+liste)	Sign.		Sign.	Bureau	
Valeur déclarée	Sign.		?	Bureau	
Signification judiciaire	Sign.		?	Bureau	
Postenveloppe+	x	x	x	Bureau	
Prime	x	x	x	Bureau	
Petit paquet / encombrant	x	x	x	Bureau	
Colis (Amazon, Redoute, QPackPlus, EPG, UPU, liste)	Sign.	Amazon	Sign.	Bureau	Station
PackUp					



Too many possible combinations

	Retour					
	Adresse incorrecte	Pas de boîte	Refusé	Parti	Décédé	Inconnu du facteur
Recommandé (+liste)	x	x	x	x	x	x
Valeur déclarée	x					
Signification judiciaire	x					
Postenveloppe+	x					
Prime	x					
Petit paquet / encombrant	x	?	?	?	?	?
Colis (Amazon, Redoute, QPackPlus, EPG, UPU, liste)	x	x	x	x	x	x
PackUp						



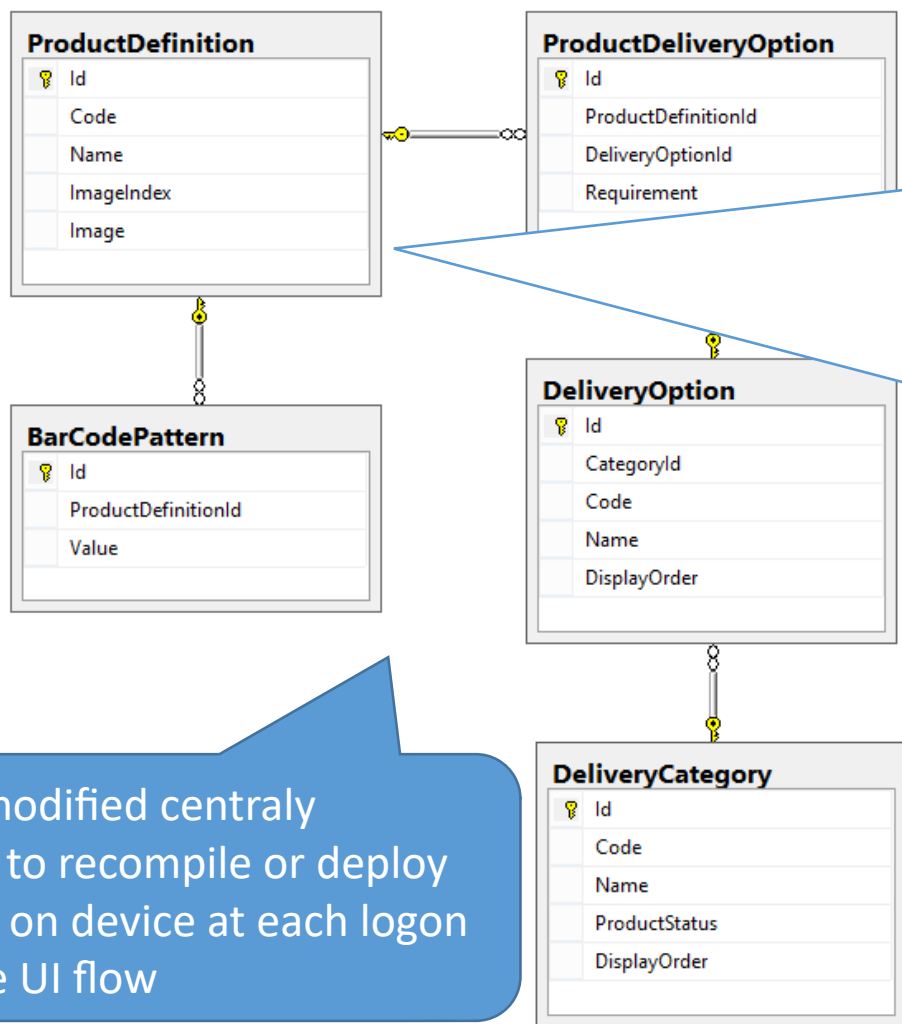
Too many different events

	Distrib.						Packup	
	Magasin fermé	Magasin congé	Dévoyé	Réexpédition	Ordre de garde	Adresse à vérifier	Dépôt	Transfer
Recommandé (+liste)	x		x	x	x	?		
Valeur déclarée	?		?	?	?	?		
Signification judiciaire	?		?	?	?	?		
Postenveloppe+						?		
Prime						?		
Petit paquet / encombrant						?		
Colis (Amazon, Redoute, QPackPlus, EPG, UPU, liste)	x		x	x	x	?		
PackUp	x	x				?	x	x



Too much complexity
in the domain

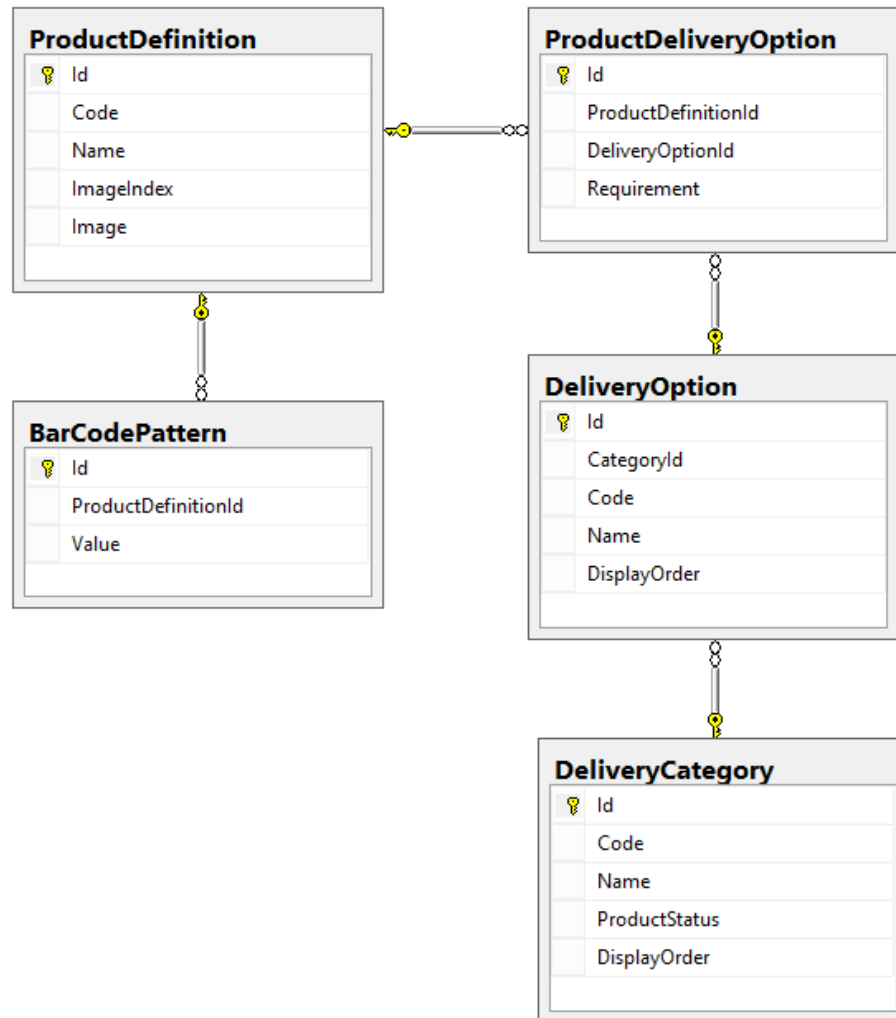
Business rules as reference data



Can be modified centrally
No need to recompile or deploy
Updated on device at each logon
Drive the UI flow

	Code	Name
1	Registered	Recommandé
2	DeclaredValue	Valeur déclarée
3	JudicialSummons	Signification judiciaire
4	PostEnvelopePlus	Postenveloppe+
5	Prime	Prime
6	SmallParcelOrCumbersome	Petit paquet/Encombr.
7	PackUp	PackUp
8	IdentifiedProvider	Colis (Redoute, etc.)
9	Amazon	Colis (Amazon)
10	QPackPlus	Colis (QPack+)
11	EPG	Colis (EPG)
12	UPU	Colis (UPU)
13	MailBox	Boîte jaune
14	RegisteredList	Liste recommandés
15	ParcelList	Liste colis

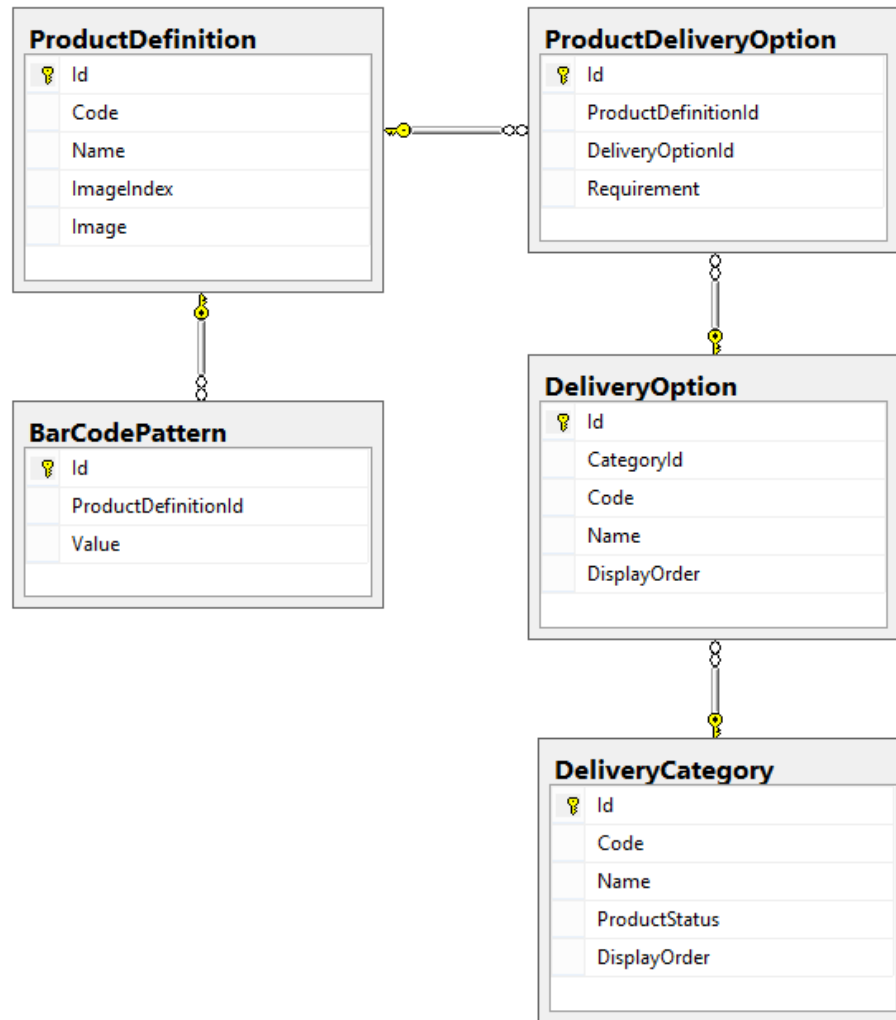
Business rules as reference data



	Code	Name	ProductStatus
1	Delivered	Remis	1
2	Notified	Avisé	2
3	ReturnedToSender	Retour	3
4	SentBackToDistribution	Divers	4
5	PackUp	PackUp	1

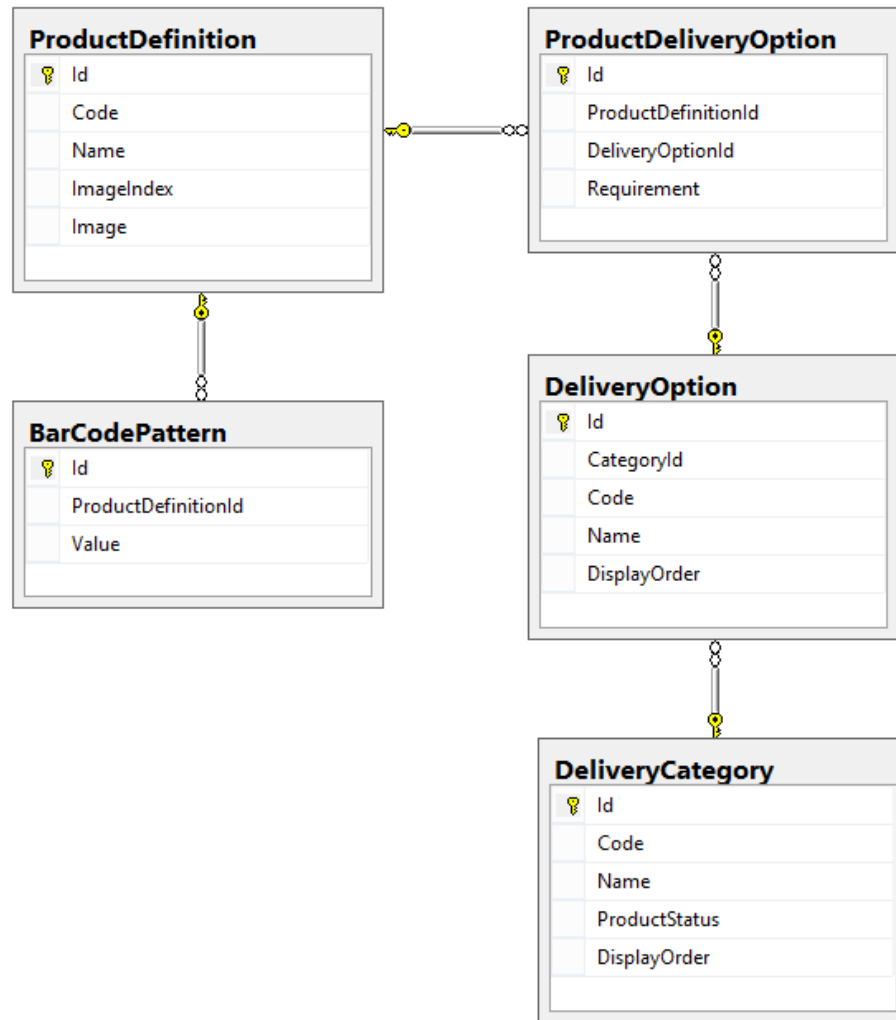
```
public enum ProductStatus
{
    Enlisted = 0,
    Delivered = 1,
    Notified = 2,
    ReturnedToSender = 3,
    SentBackToDistribution = 4,
    Collected = 5,
}
```

Business rules as reference data



	Category	Code	Name
1	Delivered	ToRecipient	Destinataire
2	Delivered	InMailBox	Boîte
3	Delivered	ToThirdParty	A un tiers
4	Notified	RecipientIsAbsent	Destinataire absent
5	Notified	PackUp2424	Pack Up 24/24
6	ReturnedToSender	IncorrectAddress	Adresse incorrecte
7	ReturnedToSender	NoMailBox	Pas de boîte
8	ReturnedToSender	Refused	Refusé
9	ReturnedToSender	Gone	Parti
10	ReturnedToSender	Deceased	Décédé
11	ReturnedToSender	UnknownByAgent	Inconnu du facteur
12	SentBackToDistribution	StoreClosed	Magasin fermé
13	SentBackToDistribution	WrongDestination	Dévoyé
14	SentBackToDistribution	Redirection	Réexpédition
15	SentBackToDistribution	Retention	Ordre de garde
16	SentBackToDistribution	AddressToBeChecked	Adresse à vérifier
17	Pack Up	Deposit	Dépôt
18	Pack Up	Transfer	Transfert
19	Pack Up	PackupSiteClosed	Site Pack Up fermé
20	Pack Up	PackupSiteOnVacation	Site Pack Up congé

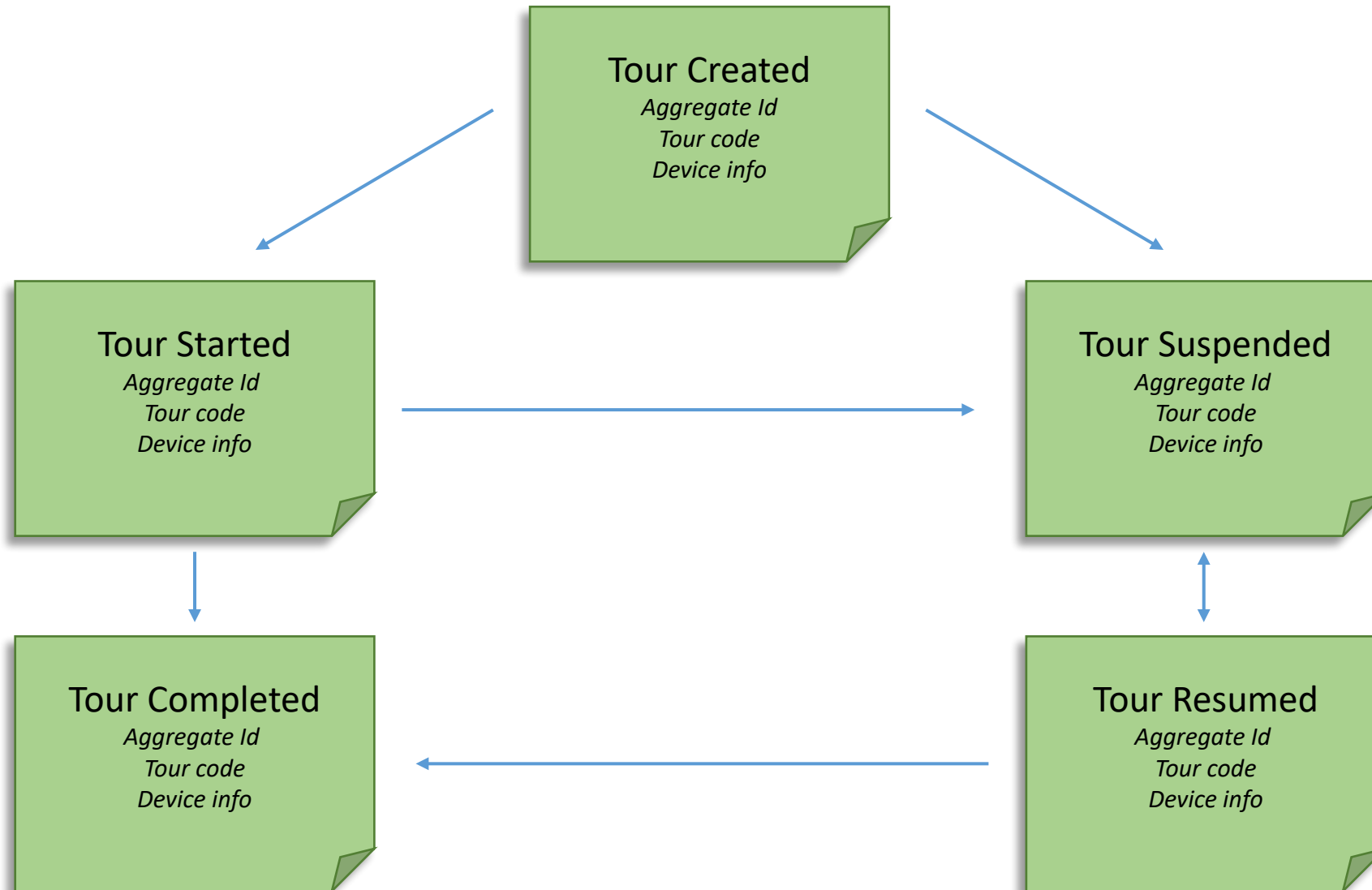
Business rules as reference data



	DeliveryCategory	DeliveryOption	ProductDefinition	Requirement
1	Delivered	InMailBox	Amazon	0
2	Delivered	InMailBox	PostEnvelopePlus	0
3	Delivered	InMailBox	Prime	0
4	Delivered	InMailBox	SmallParcelOrCumbersome	0
5	Delivered	ToRecipient	Amazon	1
6	Delivered	ToRecipient	DeclaredValue	1
7	Delivered	ToRecipient	EPG	1
8	Delivered	ToRecipient	IdentifiedProvider	1
9	Delivered	ToRecipient	JudicialSummons	1
10	Delivered	ToRecipient	ParcelList	1
11	Delivered	ToRecipient	PostEnvelopePlus	0
12	Delivered	ToRecipient	Prime	0
13	Delivered	ToRecipient	QPackPlus	1
14	Delivered	ToRecipient	Registered	1
15	Delivered	ToRecipient	RegisteredList	1

```
public enum DeliveryRequirement
{
    None = 0,
    SignatureRequired = 1,
    SignatureAndNameRequired = 2,
    NotificationOfficeRequired = 3,
    PickupStationRequired = 4,
}
```

Simplifying the events : tour status



Simplifying the events : before & during tour

Before tour

Product enlisted

Aggregate Id
Product code
Product type

Enlistment cleared

Aggregate Id
List of product codes

Enlistment reopened

Aggregate Id

During tour

Product processed

Aggregate Id
Product code
Product type
Processing status
Delivered to
Signature

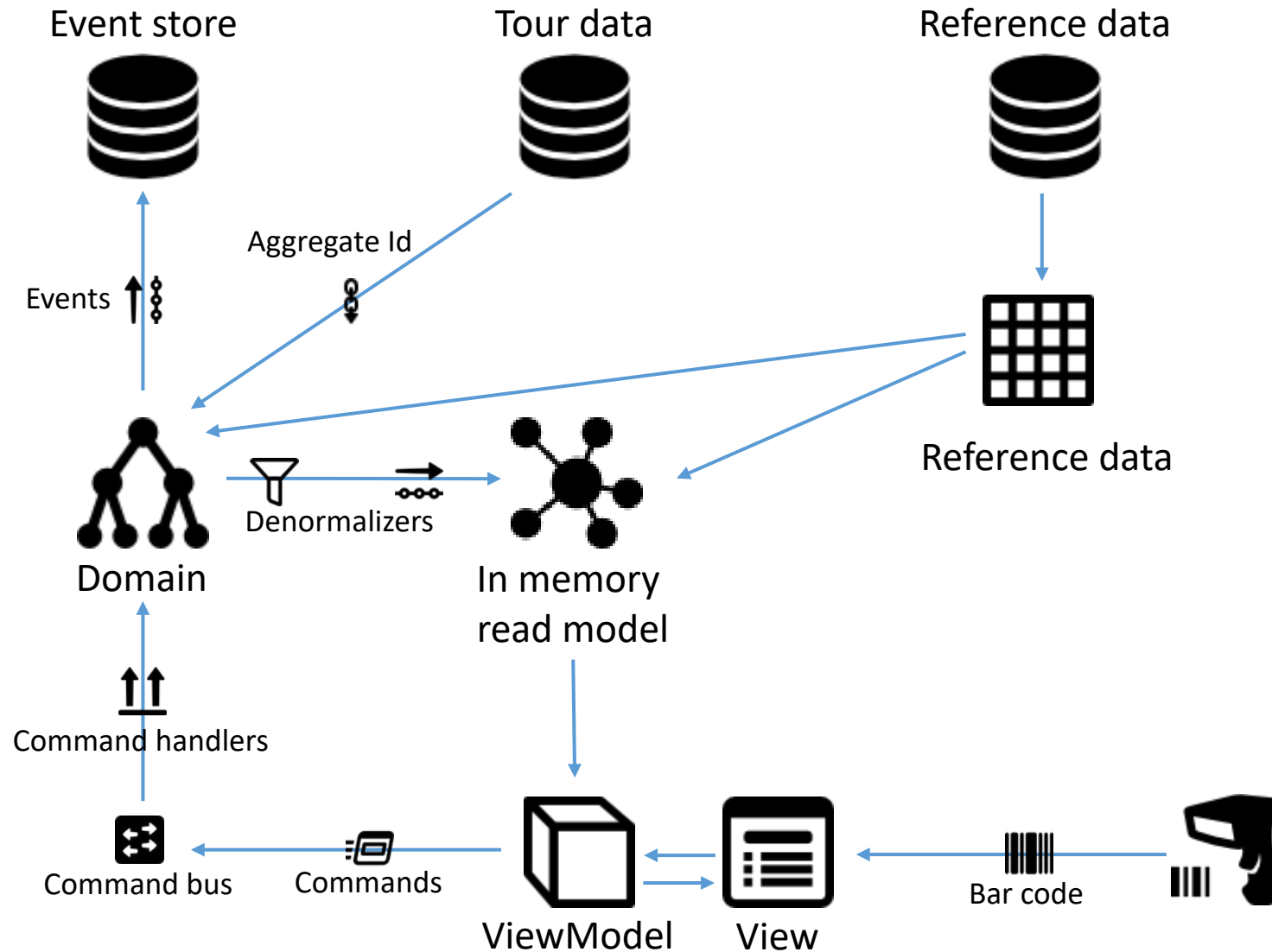
Mail box collected

Aggregate Id
Product code
Product type

What does it look like ?

Explaining the architecture on the device and on server side

Architecture on device



Aggregate with event sourcing

- *Tour* is the only aggregate
 - Encapsulate its complete life cycle
 - Keeps very little state
 - No *Product* entity in the domain
 - Just events
 - 1 aggregate == 1 stream of events
1. Check aggregate invariants
 - Verify business rules
 - Compute required data
 2. Raise event
 - Raise an event containing all required data
 - Never change state
 3. Apply event to change state
 - Use event data to change state
 - Replay when rehydrating the aggregate from store

Idempotence

$$f(f(x)) = f(x)$$

Can call aggregate methods any number of times

Can send same events any number of times

Can scan same bar code any number of times

```
[Test]
public void Start_Tour_Is_Idempotent()
{
    var tour = Create();
    tour.Start();
    Assert.DoesNotThrow(() => tour.Start());
    Assert.That(tour.GetUncommittedEvents<TourCreated>().Count, Is.EqualTo(1));
}
```

Method called twice

Only on event raised

NEventStore

```
public interface IStoreEvents : IDisposable
{
    IPersistStreams Advanced { get; }
    IEventStream CreateStream(Guid streamId);
    IEventStream OpenStream(Guid streamId, int minRevision, int maxRevision);
    IEventStream OpenStream(Snapshot snapshot, int maxRevision);
}
```

A stream is easily opened

```
public interface IEventStream : IDisposable
{
    Guid StreamId { get; }
    int StreamRevision { get; }
    int CommitSequence { get; }
    ICollection<EventMessage> CommittedEvents { get; }
    IDictionary<string, object> CommittedHeaders { get; }
    ICollection<EventMessage> UncommittedEvents { get; }
    IDictionary<string, object> UncommittedHeaders { get; }
    void Add(EventMessage uncommittedEvent);
    void CommitChanges(Guid commitId);
    void ClearChanges();
}
```

A stream is a set of ordered event messages

Event messages are saved within a stream as a series of commits

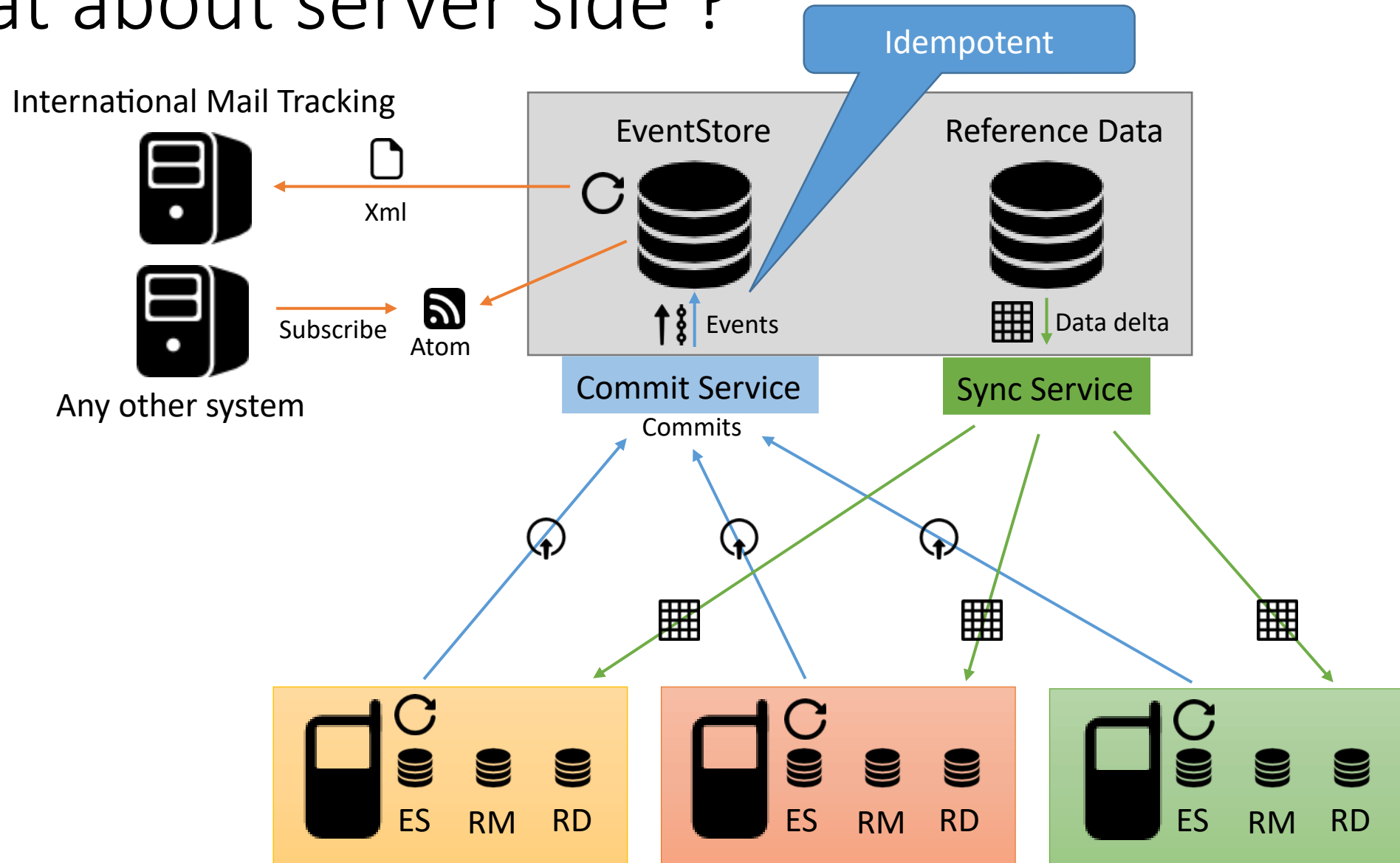
NEventStore commit

```
public Commit(  
    Guid streamId, int streamRevision, Guid commitId,  
    int commitSequence, DateTime commitStamp,  
    Dictionary<string, object> headers,  
    List<EventMessage> events)  
    : this()  
{  
    StreamId = streamId;  
    CommitId = commitId;  
    StreamRevision = streamRevision;  
    CommitSequence = commitSequence;  
    CommitStamp = commitStamp;  
    Headers = headers ?? new Dictionary<string, object>();  
    Events = events ?? new List<EventMessage>();  
}
```

	Column Name	Data Type	Allow Nulls
🔑	StreamId	uniqueidentifier	☐
	StreamRevision	int	☐
	Items	tinyint	☐
	CommitId	uniqueidentifier	☐
🔑	CommitSequence	int	☐
	CommitStamp	datetime	☐
	Dispatched	bit	☐
	Headers	varbinary(MAX)	☒
	Payload	varbinary(MAX)	☐

Commit contain many events
Commit is serializable
No snapshot used

What about server side ?



ATOM : tour feed

WebAPI

Fabrik.Common.WebAPI.AtomPub

 Subscribe to this feed using  Live Bookmarks
☐ Always use Live Bookmarks to subscribe to feeds.

ADT Tours

tours@localhost

[BS001F](#)
dimanche, le 16 novembre 2014 12:48
... Adt.MailDelivery.Domain.Tour (32 events)

[IG001F](#)
dimanche, le 16 novembre 2014 12:40
... Adt.MailDelivery.Domain.Tour (5 events)

[IG001L](#)
dimanche, le 16 novembre 2014 12:41
... Adt.MailDelivery.Domain.Tour (8 events)

Shamelessly borrowed
the idea from
@GetEventStore

ATOM : event feed

Event type & Product code

BS001F

Adt.MailDelivery.Domain.Tour (36 events)

0 - TourCreated (BS001F)

dimanche, le 16 novembre 2014 13:43

```
{ "Aggregateld": "fa27f22a-ae7-4b88-b915-32c8af9186c0", "Metadata": { "TourCode": "BS001F", "TourDate": "2014-11-16T00:00:00+01:00", "DeviceId": "001346997DC50108-90C40900000000004936303041583133534C4C42413631330000", "DeviceName": "Pocket_PC", "SoftwareVersion": "3.0.0.0" }, "Occurred": "2014-11-16T13:43:48+01:00", "EventType": "TourCreated" }
```

Event metadata

1 - ProductEnlisted (RR232770093LU)

dimanche, le 16 novembre 2014 13:44

```
{ "ProductCode": "RR232770093LU", "ProductType": { "Code": "Registered", "Name": "Recommandé" }, "Aggregateld": "fa27f22a-ae7-4b88-b915-32c8af9186c0", "Metadata": { "TourCode": "BS001F", "TourDate": "2014-11-16T00:00:00+01:00", "DeviceId": "001346997DC50108-90C40900000000004936303041583133534C4C42413631330000", "DeviceName": "Pocket_PC", "SoftwareVersion": "3.0.0.0" }, "Occurred": "2014-11-16T13:44:11+01:00", "EventType": "ProductEnlisted" }
```

Product type

24 - MailBoxCollected (B1235001)

dimanche, le 16 novembre 2014 13:45

```
{ "ProductCode": "B1235001", "ProductType": { "Code": "MailBox", "Name": "Boîte jaune" }, "Aggregateld": "fa27f22a-ae7-4b88-b915-32c8af9186c0", "Metadata": { "TourCode": "BS001F", "TourDate": "2014-11-16T00:00:00+01:00", "DeviceId": "001346997DC50108-90C40900000000004936303041583133534C4C42413631330000", "DeviceName": "Pocket_PC", "SoftwareVersion": "3.0.0.0" }, "Occurred": "2014-11-16T13:45:30+01:00", "EventType": "MailBoxCollected" }
```

25 - ProductProcessed (VV200108793LU)

dimanche, le 16 novembre 2014 13:47

```
{ "Status": { "ReasonCode": "ToThirdParty", "ReasonName": "A un tiers", "StatusCode": "Delivered", "StatusName": "Remis" }, "DeliveredTo": "JOHN DOE", "Signature": "iVBORw0KGgoAAAANSUUhEUgAAADIAAAEBCAIAAABDQlPgAAAAAXNSR0IArs4c6QAAAAARnQU1BAACxjwv8YQUAAAGY0hSTQAAeiYAAICEAAD6AAAAgOgAAHUwAADqYAAAOpgAABdwnLpRPAAAAAAlwSFlzAAAdhwAAHYcBj+XxZQAAECxJREFUeF7tm9mWW2/Pnzx/8gQAEIACBNAJf2uUPBAAAQikEfiRthMbQQACEIDArw8YoAABCEAAApKE0G4mbfaCAAQgwGuXMwABCEAgwCv3Vze7AYBCLQngHbbHwEAQAACuQTQbi5vdoMABNoTQLvtjwAAIACBNAJoN5c3u0EAAu0JoN32RwAAEIBALgG0m8ub3SAAgfYE0G77lwAA/BAAAAQikEkC7ubzZDQIQaE8A7bY/AgCAAARyCaDdXN7sBgEitCeAdtsfAQBAAAK5BNBuLm92gwAE2hNAu+2PAAAgAIFcAmg3lze7QQAC7Qmg3fZHAQAQgEauAbSby5vdlACB9gTQbvjsjAAAAIQCXANrN5c1uEIBAewJot/0RAAAEIJBLAO3m8mY3CECgPQG02/4IAAACEMglgHZzebMbBCDQngDabX8EAAABCOQSQLu5vNkNAH8oTwDttj8CAIAABHIIJoN1c3uwGAQI0J4B22x8BAEAAARKE0G4ub3aDAATaE0C77Y8AACAAgVwCaDeXN7tBAALtCaDd9kCAABCAQC4BtJvLm90gAIH2BNBu+yMAAAhAIJcA2s3IffRuP179ObpjmoC/pdAgwAEvgmgXU6CncCDW58XQR52uMw8lwDaPTfb4M7ulXq51aWg11dgAASOIYB2j4kytZGbRqd2Rb5TuBh8KgG0e2qysX19C9S2x9Qz2bYfSyCgTMB4c5RbWlib2SwLaxBcyvbUfWgE+QomS0k5BNDub86DX8cvMU5OtEG7LCSAeYMyYllxAk21OytZ53jxQzBVt/bczRWQuPryw+cuFT99Yp5hUPnfKWE2ihXf7vqSXnJsK534Vh3iUBsUgVAdrl1/XX3gQ13688OEDh4U1sxQEBAmcrF3eUAsPXNxt977InF0WYmEpCBgIHKtdnGs4DR+mRD91nz/qXVs/q0FAh8CZ2I3yaELcz+
```

Product status

Event date & type

Signature

Reference data maintenance

ASP.NET MCV 5
Scaffolding
Sync Fwk

[Accueil](#) [A propos](#) [Contact](#)



ADT - Reference Data.

Administration des données de référence ADT

Vous pouvez administrer:

- 1 Définition des produits**
Les produits qui peuvent être scannés par l'application ADT mobile dans le but de les remettre au destinataire, des les aviser en cas d'absence, etc. [Voir...](#)
- 2 Patterns de code-barre**
Les patterns exprimés sous forme d'expression régulière qui correspondent aux différents code-barre valides pour chaque définition de produit. [Voir...](#)
- 3 Catégories de livraison**
Les catégories de livraison qui correspondent aux boutons disponibles dans l'écran de tournée d'ADT [Voir...](#)
- 4 Options de livraison**
Les options de livraison pour chaque catégorie [Voir...](#)
- 5 Option de livraison par produit**
Les option des livraisons autorisées pour chaque définition de produit [Voir...](#)

CRUD



Option de livraison par produit

[Nouveau](#)

Recommandé (Registered)

Remis (Delivered)

Destinataire (ToRecipient)	SignatureRequired			
A un tiers (ToThirdParty)	SignatureAndNameRequired			

Avisé (Notified)

Destinataire absent (RecipientIsAbsent)	NotificationOfficeRequired			
---	----------------------------	--	--	--

Retour (ReturnedToSender)

Adresse incorrecte (IncorrectAddress)	None			
Pas de boîte (NoMailBox)	None			
Refusé (Refused)	None			

Don't really need
event sourcing there

CAP Theorem



Consistency → all nodes see the same data at the same time

Don't need to see the same data on all devices at the same time

Each stream of events constitutes an isolated log of tour events

Share the same reference data that is synced regularly



Availability → a guarantee that every request receives a response about whether it was successful or failed

Devices don't need server connectivity to work properly

Just need to guarantee that all events are eventually uploaded on server



Partition tolerance → the system continues to operate despite arbitrary message loss or failure of part of the system

Data is uploaded from devices to 1 server as soon as connectivity is available

Idempotence allows to send messages several times if needed

Where is my business logic ?

Designing and implementing the domain

Aggregate

Base class for
the aggregate

```
public class Tour : AggregateBase
{
    private TourMetadata _metadata;
    private TourStatus _status;
    private readonly List<string> _enlistedProducts;
    private readonly List<string> _collectedMailBoxes;
```

Very little
state

CommonDomain

```
public void Start()
{
    if (IsStarted())
        return;
    CheckCanStart();
    RaiseEvent(new TourStarted(Id, _tourCode, _device, DateTime.Now));
}
```

Idempotence

```
private bool IsStarted()
{
    return _status == TourStatus.Started;
}
```

Raising event

```
private void CheckCanStart()
{
    if (_status != TourStatus.Created)
        throw new InconsistentStateException(String.Format(Messages.CannotStartTour, _status));
}
```

```
private void Apply(TourStarted evt)
{
    _status = TourStatus.Started;
}
```

Changing state

Testing the domain

TDD

.NET 3.5 → CF

Nunit

NSubstitute

Arrange

```
public void Deliver_Product_Then_Notify()
{
    string deliveredTo = "Destinataire";
    Bitmap signature = new Bitmap(15, 15);
    string comment = "This is a comment";
    string differentComment = "This is a different comment";
    string location = "Differdange";
    byte[] imageBytes = signature.ToByteArray(ImageFormat.Png);

    Tour tour = TourDefaultValues.CreateTour()
        .ExpectMatchProduct(ProductDefinitions.Amazon.SampleCode, ProductDefinitions.Amazon.Definition)
        .ExpectProductDeliveryOption(ProductDefinitions.Amazon.Definition,
            DeliveryOptions.Delivered.ToRecipient, DeliveryRequirement.SignatureRequired)
        .ExpectProductDeliveryOption(ProductDefinitions.Amazon.Definition,
            DeliveryOptions.Notified.RecipientIsAbsent, DeliveryRequirement.NotificationOfficeRequired)
        .ExpectDeliveryCategory(DeliveryOptions.Delivered.ToRecipient, DeliveryOptions.Delivered.Category)
        .ExpectDeliveryCategory(DeliveryOptions.Notified.RecipientIsAbsent, DeliveryOptions.Notified.Category);
    tour.Start();
}
```

Test stubs

Test utility
extensions

Testing the domain (continued)

Act

```
tour.ProcessProduct(ProductDefinitions.Amazon.SampleCode, DeliveryOptions.Delivered.ToRecipient, deliveredTo, signature, null, comment);  
tour.ProcessProduct(ProductDefinitions.Amazon.SampleCode, DeliveryOptions.Notified.RecipientIsAbsent, null, null, location, differentComment);
```

Assert

```
var events = tour.GetUncommittedEvents<ProductProcessed>();  
  
Assert.That(events.Count, Is.EqualTo(2));  
ProductProcessed evt = events.Last();  
Assert.That(evt.ProductCode, Is.EqualTo(ProductDefinitions.Amazon.SampleCode));  
Assert.That(evt.Status.StatusCode, Is.EqualTo(DeliveryOptions.Notified.Category.Code));  
Assert.That(evt.Status.StatusName, Is.EqualTo(DeliveryOptions.Notified.Category.Name));  
Assert.That(evt.Status.ReasonCode, Is.EqualTo(DeliveryOptions.Notified.RecipientIsAbsent.Code));  
Assert.That(evt.Status.ReasonName, Is.EqualTo(DeliveryOptions.Notified.RecipientIsAbsent.Name));  
Assert.That(evt.DeliveredTo, Is.Null);  
Assert.That(evt.Signature, Is.Null);  
Assert.That(evt.Location, Is.EqualTo(location));  
Assert.That(evt.Comment, Is.EqualTo(differentComment));
```

Test utility
extensions

Process product in aggregate

CommonDomain

```
public void ProcessProduct(string productCode, DeliveryOption deliveryOption, string deliveredTo, Bitmap signature, string location, string comment)
{
    CheckCanProcessProduct();

    string upperCaseProductCode = productCode.ToUpper();
    ProductDefinition productDefinition = GetProductDefinition(upperCaseProductCode);
    ProductDeliveryOption productDeliveryOption = ReferenceData.ProductDeliveryOption(productDefinition, deliveryOption);
    DeliveryCategory deliveryCategory = ReferenceData.DeliveryCategory(deliveryOption);

    CheckSignature(deliveryOption, signature, productDefinition, productDeliveryOption);
    CheckDeliveredTo(deliveryOption, deliveredTo, productDefinition, productDeliveryOption);
    CheckNotificationOffice(deliveryOption, location, productDefinition, productDeliveryOption);

    ProductType type = new ProductType(productDefinition.Code, productDefinition.Name);
    ProcessingStatus status = new ProcessingStatus(deliveryCategory.Code, deliveryCategory.Name, deliveryOption.Code, deliveryOption.Name);
    string deliveredToUpper = (String.IsNullOrEmpty(deliveredTo)) ? null : deliveredTo.RemoveAccent().ToUpper();
    byte[] signatureBytes = productDeliveryOption.IsSignatureRequired() ? signature.ToByteArray(ImageFormat.Png) : null;
    string pickupStation = GetPickupStationName(productCode, productDefinition, productDeliveryOption);
    string actualLocation = location ?? pickupStation;

    RaiseEvent(new ProductProcessed(Id, _metadata, upperCaseProductCode, type, DateTime.Now,
        status, deliveredToUpper, signatureBytes, actualLocation, comment));
}
```

Check
Get
Check
Build
Raise

event data
invariants
event data
event

ref data
invariants
event data
event

Apply actually
does nothing

Reference data repository

```
public interface IStoreReferenceData
{
    void Clear();

    ProductDefinition
    DeliveryCateg
    ProductDefinition
    {
        IEnumerable<ProductDefinition> ProductDefinitions();
        ProductDefinition ProductDefinition(string code);
        ProductDefinition MatchProduct(string productCode);
    }

    DeliveryCateg
    DeliveryCateg
    {
        IEnumerable<DeliveryCategory> DeliveryCategories();
        DeliveryCategory DeliveryCategory(DeliveryOption option);
        DeliveryCategory DeliveryCategory(string code);
    }

    DeliveryOptions
    DeliveryOptions
    {
        IEnumerable<DeliveryOption> DeliveryOptions();
        IEnumerable<DeliveryOption> DeliveryOptions(string categoryCode);
    }

    ProductDeliveryOptions
    ProductDeliveryOptions
    {
        IEnumerable<ProductDeliveryOption> ProductDeliveryOptions();
        IEnumerable<ProductDeliveryOption> ProductDeliveryOptions(string deliveryOptionCode);
        IEnumerable<ProductDeliveryOption> ProductDeliveryOptions(string productDefinitionCode, string deliveryCategoryCode);
        ProductDeliveryOption ProductDeliveryOption(ProductDefinition productDefinition, DeliveryOption deliveryOption);
    }

    NotificationOffices
    PickupStation
    {
        IEnumerable<NotificationOffice> NotificationOffices(string tourCode);
        PickupStation PickupStation(int postalCode);
    }
}
```

Reference Data

OpenNETCF.ORM

```
[Entity(KeyScheme = KeyScheme.GUID)]
public class ProductDeliveryOption
{
    public static ProductDeliveryOption Create(ProductDefinition productDefinition, DeliveryOption deliveryOption, DeliveryRequirement requirement)
    {
        return new ProductDeliveryOption { Id = Guid.NewGuid(), ProductDefinitionId = productDefinition.Id,
            DeliveryOptionId = deliveryOption.Id, Requirement = requirement };
    }
}
```

Attribute to mark
persisted field

```
[Field(IsPrimaryKey = true)]
public Guid Id { get; set; }
[Field]
public Guid ProductDefinitionId { get; set; }
[Field]
public Guid DeliveryOptionId { get; set; }

public DeliveryRequirement Requirement { get; set; }
```

Trick for enum

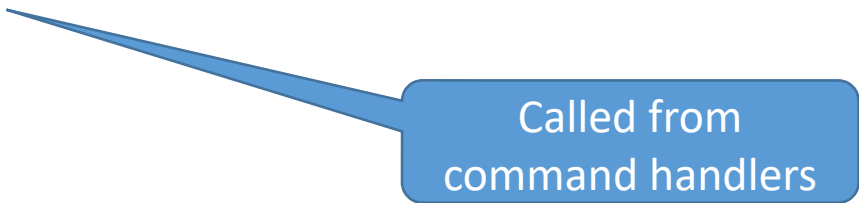
```
[Field(FieldName = "Requirement")]
public int RequirementValue
{
    get { return (int)Requirement; }
    set { Requirement = (DeliveryRequirement)value; }
}
```

```
public enum DeliveryRequirement
{
    None = 0,
    SignatureRequired = 1,
    SignatureAndNameRequired = 2,
    NotificationOfficeRequired = 3,
    PackupStationRequired = 4,
}
```

Persisting aggregate

CommonDomain

```
public interface IRepository
{
    void Clear();
    TAggregate GetById<TAggregate>(Guid id) where TAggregate : class, IAggregate;
    TAggregate GetById<TAggregate>(Guid id, int version) where TAggregate : class, IAggregate;
    void Save(IAggregate aggregate, Guid commitId, Action<IDictionary<string, object>> updateHeaders);
}
```



Called from
command handlers

NEventStore + CommonDomain → .NET CF

- Port for .NET Compact Framework

- TimerDispatchScheduler
- IDispatchCommits → bool

```
public interface IDispatchCommits : IDisposable
{
    bool Dispatch(Commit commit);
}
```

- System.CF

- Lazy<T>
- System.Transactions
- System.Collections.Concurrent
- System.Threading.Tasks



Meet the rest of my new friends



GitHub



Codeplex



ILSpy



Mono.Cecil

But now .NET is
open source

Persistence wireup



```
public IStoreEvents BuildEventStore()
{
    return Wireup.Init()
        .UsingSQLitePersistence(_settings)
        .WithDialect(new NEventStore.Persistence.SqlPersistence.SqlDialects.SqliteDialect())
        .InitializeStorageEngine()
        .UsingJsonSerialization()
        .Compress()
        .EncryptWith(Encryption.Key)
        .HookIntoPipelineUsing(RootWorkItem.Services.Get<IDenormalizeReadModels>())
        .Build();
}
```

OpenNETCF.ioC
Service Locator



Encryption



Pipeline hook to denormalize



Json serialization

Specifications for enlist in tour

.NET 3.5 → CF
SpecFlow
Gherkin

Feature: Enlist Product

In order to specify that a product will be part of the tour before it starts
As a mail delivery agent
I want to enlist the product

Scenario: Enlist product of type Registered

Given the tour 'BT121F' is created
When I scan a product bar code 'RR123456789LU'
Then a product of type 'REG' is enlisted in the tour

Scenario: Enlist product of type Prime

Given the tour 'BT121F' is created
When I scan a product bar code 'LY123456789SK'
Then a product of type 'PRI' is enlisted in the tour

Scenario: Does not enlist an unknown product type

Given the tour 'BT121F' is created
When I scan a product bar code 'xxxxxxxxxxxxxxx'
Then I get notified that the bar code is not valid

```
[Given("the tour (.*) is created")]
public void GivenTheTourIsCreated(string tourCode)
{
    ScenarioContext.Current.Pending();
}

[When("I scan a product bar code (.*)")]
public void WhenIScanAProductBarCode(string productBarCode)
{
    ScenarioContext.Current.Pending();
}

[Then("a product of type (.*) is enlisted in the tour")]
public void ThenAProductOfTypesEnlistedInTheTour(string productType)
{
    ScenarioContext.Current.Pending();
}
```

Testing on device

.NET CF
NUnitLite

```
[Test]
public void Save_And_Restore_ProductDelivered()
{
    ProcessingStatus status = new ProcessingStatus("Delivered", "Remis", "ToRecipient", "Destinataire");
    string deliveredTo = "Destinataire";
    byte[] signature = new byte[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 };
    string location = "Differdange";
    string comment = "This is a comment";

    ProductProcessed evt = new ProductProcessed(AggregateId, ProductCode, new ProductType(ProductTypeCode, ProductTypeName),
        DateTime.Now, status, deliveredTo, signature, location, comment);
    Save_And_Restore_ProductEvent(evt);

    Assert.That(evt.Status.StatusCode, Is.EqualTo(status.StatusCode));
    Assert.That(evt.Status.StatusName, Is.EqualTo(status.StatusName));
    Assert.That(evt.Status.ReasonCode, Is.EqualTo(status.ReasonCode));
    Assert.That(evt.Status.ReasonName, Is.EqualTo(status.ReasonName));
    Assert.That(evt.DeliveredTo, Is.EqualTo(deliveredTo));
    Assert.That(evt.Signature, Is.EqualTo(signature));
    Assert.That(evt.Location, Is.EqualTo(location));
    Assert.That(evt.Comment, Is.EqualTo(comment));
}
```

Saving and restoring the ProductProcessed event from the actual SQLite store

Running the tests on device

.NET CF
NUnitLite

```
public class Program
{
    static void Main(string[] args)
    {
        string codeBase = Assembly.GetExecutingAssembly().GetName().CodeBase;
        string output = Path.ChangeExtension(codeBase, "txt");
        var arguments = new List<string>();
        arguments.Add("-result:" + codeBase.Replace(".exe", "-result.xml"));
        arguments.Add("-out:" + output);
        arguments.Add("-exclude:" + TestCategories.LongRunning + "," + TestCategories.TimeDependent);

        new TextUI().Execute(arguments.ToArray());

        Process.Start(output, String.Empty);
    }
}
```



Thank you
NUnitLite

How do I interact with the system ?

Sending commands

Sending commands

```
public interface ISendCommands
{
    void Send<TCommand>(TCommand command) where TCommand : Command;
}
```

Generic

```
private void SendLoginCommand()
{
    _commandBus.Send(new LoginCommand(_login));
}
```

Concrete command

```
private void SendCollectMailCommand(string productCode)
{
    _commandBus.Send(new CollectMailBoxCommand(_readModelRepository.Current.AggregateId, productCode));
}
```

Concrete command

Handling commands

```
public interface IHandleCommand<TCommand> where TCommand : Command
{
    void Handle(TCommand command);
}
```

Implement generic interface

```
public class CollectMailBoxCommandHandler : IHandleCommand<CollectMailBoxCommand>
{
    public void Handle(CollectMailBoxCommand command)
    {
        Tour tour = _repository.GetById<Tour>(command.AggregateId);
        Ensure.NotNull(tour, String.Format(Messages.TourNotFound, command.AggregateId));

        tour.Start();
        tour.CollectMailBox(command.MailBoxCode);

        _repository.Save(tour, Guid.NewGuid());
    }
}
```

Implement handle method

Registering handlers

```
class AtStartup : IRegisterComponents
```

```
{  
    private readonly IRegisterHandlers _commandBus;
```

```
    [InjectionConstructor]
```

OpenNETCF.ioC injection

```
    public AtStartup(  
        [ServiceDependency] IRegisterHandlers commandBus,
```

```
        [ServiceDependency] IBootstrapModules bootstrapper)  
    {
```

```
        _commandBus = commandBus;
```

```
        bootstrapper.RegisterBootstrap(ModuleNames.MailDelivery, Bootstrap);
```

```
        bootstrapper.RegistrerCleanup(ModuleNames.MailDelivery, Cleanup);  
    }
```

```
    private void Bootstrap()  
    {
```

Register at bootstrap

```
        _commandBus.RegisterHandler(  
            RootWorkItem.Services.GetOrCreate<LoginCommandHandler>());
```

```
        _commandBus.RegisterHandler(  
            RootWorkItem.Services.GetOrCreate<CollectMailBoxCommandHandler>());
```

```
        _commandBus.RegisterHandler(  
            RootWorkItem.Services.GetOrCreate<ProcessProductsCommandHandler>());
```

```
        _commandBus.RegisterHandler(  
            RootWorkItem.Services.GetOrCreate<ProcessProductsCommandHandler>());  
    }
```

```
    private void Cleanup()  
    {
```

```
        _commandBus.UnregisterHandlers<LoginCommand>();
```

```
        _commandBus.UnregisterHandlers<CollectMailBoxCommand>();
```

```
        _commandBus.UnregisterHandlers<ProcessProductsCommand>();  
    }
```

Unregister at cleanup

Registering handlers in command bus

```
public interface IRegisterHandlers
```

```
{
```

```
void RegisterPreExecutionHandler(IHandleCommand<Command> handler);
```

```
void RegisterHandler<TCommand>(IHandleCommand<TCommand> handler) where TCommand : Command;
```

```
void RegisterPostExecutionHandler(IHandleCommand<Command> handler);
```

```
void UnregisterHandlers<TCommand>() where TCommand : Command;
```

```
}
```

All commands

Specific command

All commands

Sending commands with command bus

```
public class CommandBus : ISendCommands, IRegisterHandlers
```

```
{
```

```
    public void Send<TCommand>(TCommand command) where TCommand : Command
```

```
    {
```

```
        CheckHandlersExist<TCommand>();
```

```
        CallPreExecutionHandlers(command);
```

```
        CallHandlers(command);
```

```
        CallPostExecutionHandlers(command);
```

```
    }
```

```
}
```



Execute all handlers

What do I see on screen ?

Building a read model in memory

Denormalizing events

```
public interface IDenormalizeDomainEvent<TEvent>
{
    void Denormalize(TEvent @event);
}
```

Implement generic interface

```
public class ProductProcessedDenormalizer : IDenormalizeDomainEvent<ProductProcessed>
{
    public void Denormalize(ProductProcessed @event)
    {
        TourReadModel readModel = GetTourReadModel(@event.AggregateId);
        ProductDefinition definition = GetProductDefinition(@event.ProductType.Code);
        ProductStatus status = GetProductStatus(@event.Status.StatusCode);

        readModel.ProductType(definition)
            .Product(@event.ProductCode)
            .Process(status, @event.Occurred, @event.Status.ReasonName, @event.DeliveredTo, @event.Location, @event.Comment);
    }
}
```

Use event data to
update read model

PipelineHook

NEventStore

```
public interface IDenormalizeReadModels : IPipelineHook
{
    void Clear();
    void Register<TEvent>(Action<TEvent> action);
}
```

```
public class DenormalizerPipelineHook : IDenormalizeReadModels
{
    private Dictionary<Type, Action<object>> _denormalizers;
    private List<Guid> _denormalized;
```

```
    public void PostCommit(Commit committed)
    {
        Denormalize(committed);
    }
```

Denormalize when
saving events

```
    public Commit Select(Commit committed)
    {
        Denormalize(committed);
        return committed;
    }
```

Denormalize when
reloading events

```
    private void Denormalize(Commit commit)
    {
        if (_denormalized.Contains(commit.CommitId))
            return;

        Denormalize(commit.Events);
        _denormalized.Add(commit.CommitId);
    }
```

```
    private void Denormalize(IEnumerable<EventMessage> events)
    {
        foreach (var eventMessage in events)
        {
            var action = GetDenormalizerAction(eventMessage.Body);
            if (action == null)
                continue;
            action(eventMessage.Body);
        }
    }
```

Actions to
denormalize

```
    public void Register<TEvent>(Action<TEvent> action)
    {
        _denormalizers.Add(typeof(TEvent), obj => action((TEvent)obj));
    }
```

Registering denormalizers

OpenNETCF.IoC

```
class AtStartup : IRegisterComponents
{
    public void Run()
    {
        RegisterDenormalizers();
    }

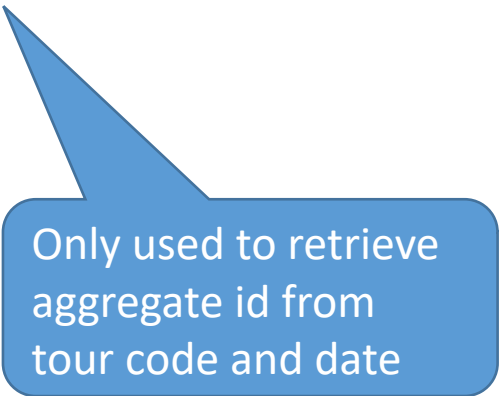
    private void RegisterDenormalizers()
    {
        RootWorkItem.Services.AddNew<ReadModels.Denormalizers.ClearedProductsFromTourDenormalizer>();
        RootWorkItem.Services.AddNew<ReadModels.Denormalizers.MailBoxCollectedDenormalizer>();
        RootWorkItem.Services.AddNew<ReadModels.Denormalizers.ProductProcessedDenormalizer>();
        RootWorkItem.Services.AddNew<ReadModels.Denormalizers.ProductRegisteredInTourDenormalizer>();
        RootWorkItem.Services.AddNew<ReadModels.Denormalizers.TourStatusDenormalizer>();
    }
}
```



Add denormalizers as
services

Persisted Read Model

```
public interface IStoreReadModels
{
    TourReadModel Current { get; }
    bool IsLoaded { get; }
    void Create(Guid aggregatId, string tourCode, DateTime dateTime);
    Guid? GetTourId(string tourCode, DateTime dateTime);
    void Clear();
}
```



Only used to retrieve
aggregate id from
tour code and date

OpenNETCF.ORM

```
[Entity(KeyScheme.GUID, NameInStore = "Tour")]
public class TourData
{
    public static TourData Create(Guid aggregatId, string tourCode, DateTime date)
    {
        return new TourData {
            AggregatId = aggregatId,
            Code = tourCode,
            DateTime = date.Date };
    }

    [Field(IsPrimaryKey = true)]
    public Guid AggregatId { get; set; }
    [Field]
    public string Code { get; set; }
    [Field]
    public DateTime DateTime { get; set; }
}
```

How do I make it pretty ?

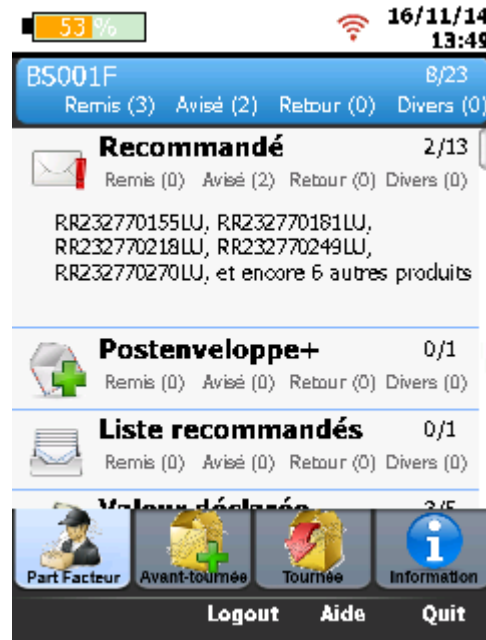
Making things look nice on screen

Views

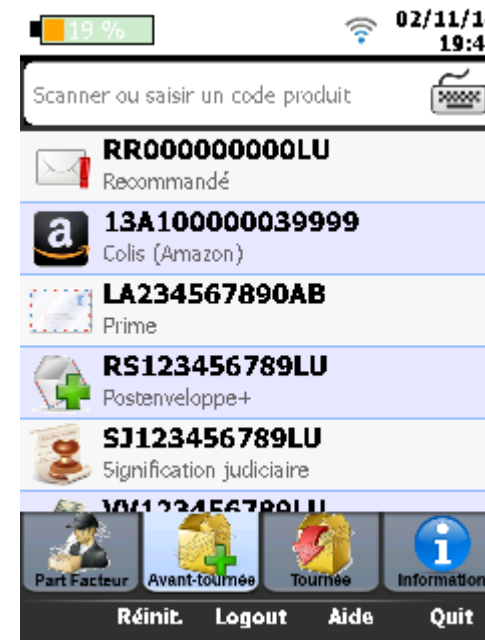
Resco MobileForms Toolkit 2014



Dock



Summary



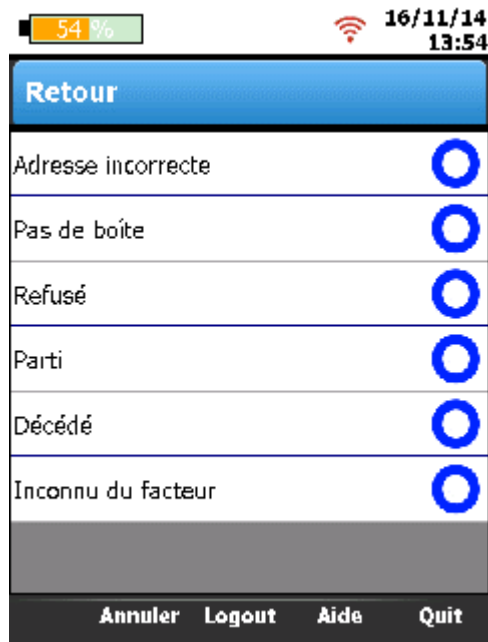
Enlisting



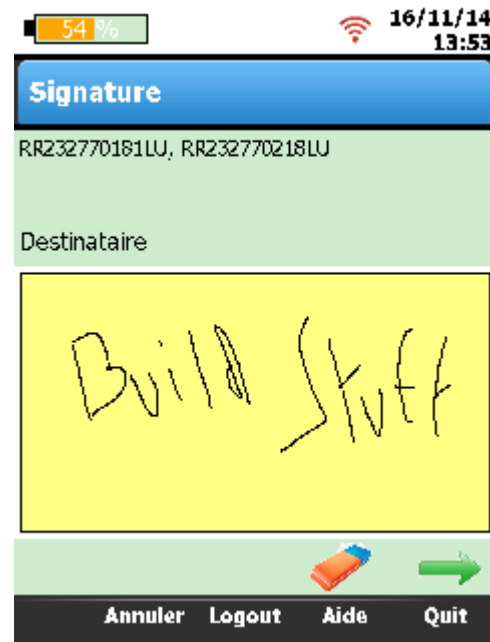
Processing

Views (continued)

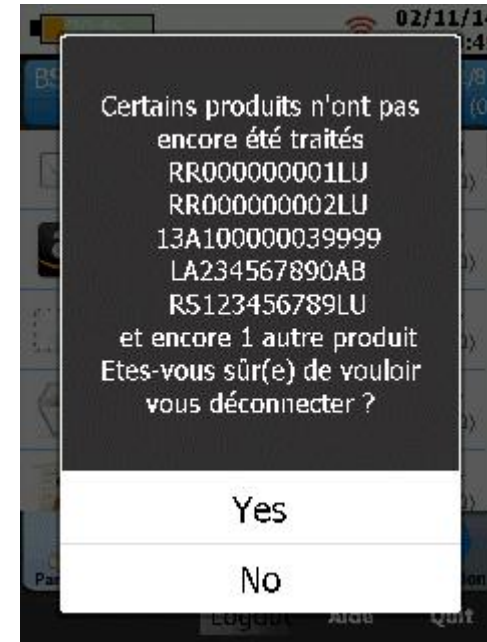
Resco MobileForms Toolkit 2014



Delivery option



Signature



Message



Synchronizing

Registering views

OpenNETCF.IoC.UI

```
class AtStartup : IRegisterComponents
```

```
{  
    public void Run()  
    {  
        RootWorkItem.SmartParts.AddNew<DockView>(ViewNames.Dock);  
        RootWorkItem.SmartParts.AddNew<DefaultView>(ViewNames.Default);  
    }  
}
```

Views are considered as SmartParts

Constants to identify view uniquely

```
class AtStartup : IRegisterComponents
```

```
{  
    public void Run()  
    {  
        //Order is important here since dependencies exist between components  
        RootWorkItem.SmartParts.AddNew<HomeView>(ViewNames.MailDelivery.Home);  
        RootWorkItem.SmartParts.AddNew<PreTourView>(ViewNames.MailDelivery.PreTour);  
        RootWorkItem.SmartParts.AddNew<TourView>(ViewNames.MailDelivery.Tour);  
        RootWorkItem.SmartParts.AddNew<InfoView>(ViewNames.MailDelivery.Info);  
        RootWorkItem.SmartParts.AddNew<MainView>(ViewNames.MailDelivery.Main);  
    }  
}
```

Constants to identify module views

Registering controls

OpenNETCF.IoC.UI

```
class AtStartup : IRegisterComponents
```

```
{
```

```
    public void Run()
```

```
    {
```

```
        RootWorkItem.SmartParts.AddNew<ScanControl>(ControlNames.Login);
```

```
        RootWorkItem.SmartParts.AddNew<ScanControl>(ControlNames.TourScan);
```

```
        RootWorkItem.SmartParts.AddNew<ScanControl>(ControlNames.DeliveryScan);
```

```
        RootWorkItem.SmartParts.AddNew<ServerConnectivityControl>(ControlNames.ServerConnectivity);
```

```
        RootWorkItem.SmartParts.AddNew<BluetoothConnectivity>(ControlNames.BlueToothConnectivity);
```

```
        RootWorkItem.SmartParts.AddNew<DateTimeControl>(ControlNames.DateTime);
```

```
        RootWorkItem.SmartParts.AddNew<DateTimeControl>(ControlNames.DateTimeDock);
```

```
        RootWorkItem.SmartParts.AddNew<BatteryControl>(ControlNames.Battery);
```

```
        RootWorkItem.SmartParts.AddNew<BatteryControl>(ControlNames.BatteryDock);
```

```
        RootWorkItem.SmartParts.AddNew<GpsControl>(ControlNames.Gps);
```

```
        RootWorkItem.SmartParts.AddNew<GprsControl>(ControlNames.Gprs);
```

```
        RootWorkItem.SmartParts.AddNew<ToolBarControl>(ControlNames.DefaultToolbar).Initialize(ControlNames.DefaultToolbar);
```

```
        RootWorkItem.SmartParts.AddNew<OptionListControl>(ControlNames.OptionList);
```

```
        RootWorkItem.SmartParts.AddNew<SignatureControl>(ControlNames.Signature);
```

```
        RootWorkItem.SmartParts.AddNew<NotificationOfficeListControl>(ControlNames.NotificationOffice);
```

```
        RootWorkItem.SmartParts.AddNew<HelpControl>(ControlNames.Help);
```

```
    }
```

Constants to identify controls uniquely

Specific initialization

Workspaces

```
public partial class DockView : SmartPart  
{
```

```
    [InjectionConstructor]
```

```
    public DockView(  
        [CreateNew] DockViewModel viewModel,  
        [ServiceDependency] INavigateViews navigator,  
        [ServiceDependency] IEventAggregator aggregator)  
        : this()  
    {
```

```
        _scanControl = _scanLoginWorkspace.Register(RootWorkItem.SmartParts[ControlNames.Login]);  
        ((IValidateInput)_scanControl).Initialize(Messages.LoginInvite, true, Login);  
  
        _wirelessStrength = _wirelessStrengthWorkspace.Register(RootWorkItem.SmartParts[ControlNames.WirelessStrength  
            + WirelessStrengthImageSize.Large]);  
        _serverConnectivity = _connectivityWorkspace.Register(RootWorkItem.SmartParts[ControlNames.ServerConnectivity]);  
        _dateTimeWorkspace.Register(RootWorkItem.SmartParts[ControlNames.DateTimeDock]);  
        _batteryWorkspace.Register(RootWorkItem.SmartParts[ControlNames.BatteryDock]);  
    }
```

```
    public override void OnActivated()  
    {  
        _scanControl.OnActivated();  
        _wirelessStrength.OnActivated();  
        _serverConnectivity.OnActivated();  
    }
```

View inherit from SmartPart

OpenNETCF.IoC.UI

Use the constants to retrieve the control instance

A workspace is a visual placeholder where a control is registered

Event aggregator

Not a domain event,
but UI events

Micro.EventAggregator

```
public interface IHandleEvent {}

public interface IHandleEvent<TMessage> : IHandleEvent
{
    void Handle(TMessage message);
}

public interface IEventAggregator
{
    Action<Action> PublicationThreadMarshaller { get; set; }

    void Subscribe(object instance);
    void Unsubscribe(object instance);
    void Publish(object message);
    void Publish(object message, Action<Action> marshal);
}
```

A SmartPart can handle
several events

```
public class RegisterUICommandEvent
{
    public RegisterUICommandEvent(string toolbarName, IUICommand command)
    {
        ToolbarName = toolbarName;
        Command = command;
    }

    public string ToolbarName { get; private set; }
    public IUICommand Command { get; private set; }
}

public partial class ToolbarControl : SmartPart,
    IHandleEvent<RegisterUICommandEvent>,
    IHandleEvent<RaiseCanExecuteEvent>
{
    public new void Handle(RegisterUICommandEvent message) {}
    public new void Handle(RaiseCanExecuteEvent message) {}
}
```

What about low level stuff ?

Getting closer to the bare metal

Bar code patterns

WindowsCE MessageWindow
Handheld.NAUTIZ API
Regex

```
public interface IMonitorBarcodes : IDisposable
{
    void Register(IHandleBarcodes barCodeHandler);
    void Unregister(IHandleBarcodes barCodeHandler);
}
```

Read bar code from scanner and publish it

Register / unregister handler

```
public interface IHandleBarcodes
{
    void SetData(string barcode);
}
```

To be implemented by the bar code consumer

```
^(?<year>\d{2})(?<ip>(?!A)(\w{2}))(?<number>\d{6})(?<code>[9])(?<postalcode>\d{4})$
(^R(?!S)([A-Z])(\d{9})([A-Z]{2})$)|(^RS(\d{9})SK$)
^RS(\w{9})LU$
^B(?<postalcode>\d{4})(?<serial>\d{3})$
```

Regex patterns to match bar codes

Testing bar code patterns

```
[Test]
public void Amazon()
{
    //Amazon product
    AssertAllPatterns(true, "13A100000039999", ProductDefinitions.Amazon.Pattern);
    AssertAllPatternsBut(false, "13A100000039999", ProductDefinitions.Amazon.Pattern);

    //Amazon pickup
    AssertAllPatterns(false, "13A100000080999", ProductDefinitions.Amazon.Pattern);

    AssertAllPatterns(false, "X3A100000039999", ProductDefinitions.Amazon.Pattern);
    AssertAllPatterns(false, "1XA100000039999", ProductDefinitions.Amazon.Pattern);
    AssertAllPatterns(false, "13B100000039999", ProductDefinitions.Amazon.Pattern);
    AssertAllPatterns(false, "130100000039999", ProductDefinitions.Amazon.Pattern);
    AssertAllPatterns(false, "13A1000000A9999", ProductDefinitions.Amazon.Pattern);
    AssertAllPatterns(false, "13A100000009999", ProductDefinitions.Amazon.Pattern);
    AssertAllPatterns(false, "13A10000003X999", ProductDefinitions.Amazon.Pattern);
    AssertAllPatterns(false, "13A100000039X99", ProductDefinitions.Amazon.Pattern);
    AssertAllPatterns(false, "13A1000000399X9", ProductDefinitions.Amazon.Pattern);
    AssertAllPatterns(false, "13A10000003999X", ProductDefinitions.Amazon.Pattern);
    AssertAllPatterns(false, "13A1000000399990", ProductDefinitions.Amazon.Pattern);
    AssertAllPatterns(false, "13A10000003999", ProductDefinitions.Amazon.Pattern);
    AssertAllPatterns(false, "13A10000039999", ProductDefinitions.Amazon.Pattern);
    AssertAllPatterns(false, "13A1000000039999", ProductDefinitions.Amazon.Pattern);
}
```

TDD
.NET 3.5 → CF
NUnit

```
private void AssertAllPatterns(bool expected, string value,
                               params string[] patterns)
{
    foreach (var pattern in patterns)
    {
        Assert.AreEqual(
            expected,
            new Regex(pattern).IsMatch(value),
            String.Format("{0}' {1}' {2}' : {3}",
                           value,
                           expected ? "does not match" : "matches",
                           Patterns[pattern], pattern
            )
        );
    }
}
```

WiFi management

```
public void EnableWireless()
{
    if (IsWirelessEnabled)
        return;
    this.Log().Info(InfrastructureMessages.SwitchingWirelessOn);
    try
    {
        var radio = GetRadio();
        radio.RadioState = RadioState.On;
    }
    catch (Exception e)
    {
        this.Log().Error(InfrastructureMessages.WirelessError, e);
        throw;
    }
}
```

OpenNETCF.WindowsMobile

```
private IRadio GetRadio()
{
    try
    {
        return Radios.GetRadios().FirstOrDefault(r => r.RadioType == RadioType.WiFi);
    }
    catch (Exception)
    {
        throw new NetworkException(InfrastructureMessages.NoWirelessRadioFound);
    }
}
```

Vibrate

OpenNETCF.WindowsCE

```
public void Vibrate(int milliseconds)
{
    ThreadPool.QueueUserWorkItem(new WaitCallback(x =>
    {
        OpenNETCF.WindowsCE.Notification.Led vib = new OpenNETCF.WindowsCE.Notification.Led();
        try
        {
            vib.SetLedStatus(3, OpenNETCF.WindowsCE.Notification.Led.LedState.On);
            System.Threading.Thread.Sleep(milliseconds);
        }
        finally
        {
            vib.SetLedStatus(3, OpenNETCF.WindowsCE.Notification.Led.LedState.Off);
        }
    }));
}
```

Use ThreadPool
whenever you can

WTF ???
LED == vibrate

So what ?

Bringing it all together

Advantages of events



Scalability

Events are very simple and self contained

Server receives events when device has connectivity



Concurrency

Never need to access the events from other devices

Except reference data, everything is written only once



Simplicity

Only one aggregate

The rest is only events



Testability

Unit tests are easy to write

BDD can be used

Pay attention



Wording and naming is essential

Tense is important → events are in the past tense

Difficult to change events names once in production



Know your platform limitations

Threading, memory, etc.

Libraries and framework availability

Expect the dreaded NotImplementedException from .Net CF

Get ready to draw stuff yourself if you like it nice



Low level stuff is tricky

Use libraries and be clever about it

Get familiar with native code / Pinvoke <http://www.pinvoke.net>

TDD helps a great deal

Tips & tricks about .NET CF



First thing to do when working with CF in VS2008

C:\Windows\Microsoft.NET\Framework\v3.5\Microsoft.CompactFramework.Common.targets

```
<Target
  Name="PlatformVerificationTask" Condition="'$(Configuration)' != 'Debug'">
  <PlatformVerificationTask
    PlatformFamilyName="$(PlatformFamilyName)"
    PlatformID="$(PlatformID)"
    SourceAssembly="@{(IntermediateAssembly)"
    ReferencePath="@{(ReferencePath)"
    TreatWarningsAsErrors="$(TreatWarningsAsErrors)"
    PlatformVersion="$(TargetFrameworkVersion)"/>
  </Target>
```

This stuff will save you
some precious time



Follow the white rabbit

Get ready to dig deep and get your hands dirty

[What are the most valuable .Net Compact Framework Tips, Tricks, and Gotcha-Avoiders?](#)

Links

- My GitHub
 - <https://github.com/cpontet>
- NEventStore
 - <http://neventstore.org>
 - <https://github.com/NEventStore>
- SQLite
 - [System.Data.Sqlite](http://system.data.sqlite)
- OpenNETCF
 - <http://www.opennetcf.com>
 - <https://ioc.codeplex.com>
- Resco MobileForms Toolkit
 - <http://www.resco.net>
- Nunit
 - <http://www.nunit.org>
 - <http://nunitlite.org>
- SpecFlow
 - <http://www.specflow.org>
- Thanks to Icon8 for the icons
 - <http://icons8.com>

Thank you for coming



Special thanks to



agilepartner

Don't forget

#play14

March 27-29 2015