

HM Government

Modelling Communications

xtUML Day 2019

© Crown Copyright 2019

HM Government

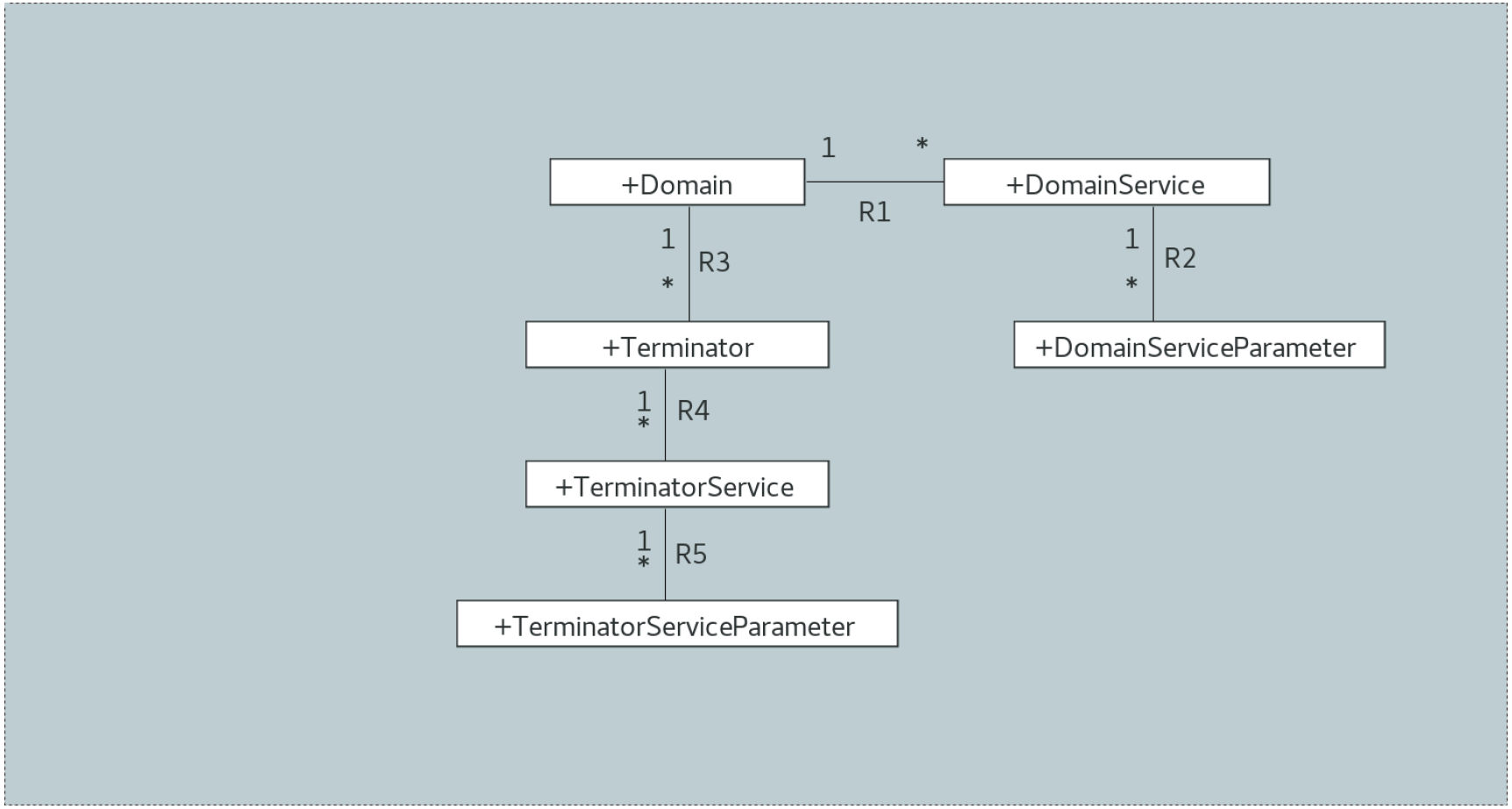
Our Software

- Large distributed command and control system
- Low bandwidth, high latency WAN communications
- Proprietary message middleware: ‘Message Bus’
- Mixture of MASL (generated to C++) and hand-coded C++ and Java
- All message interfaces generated from MASL.

© Crown Copyright 2019

HM Government

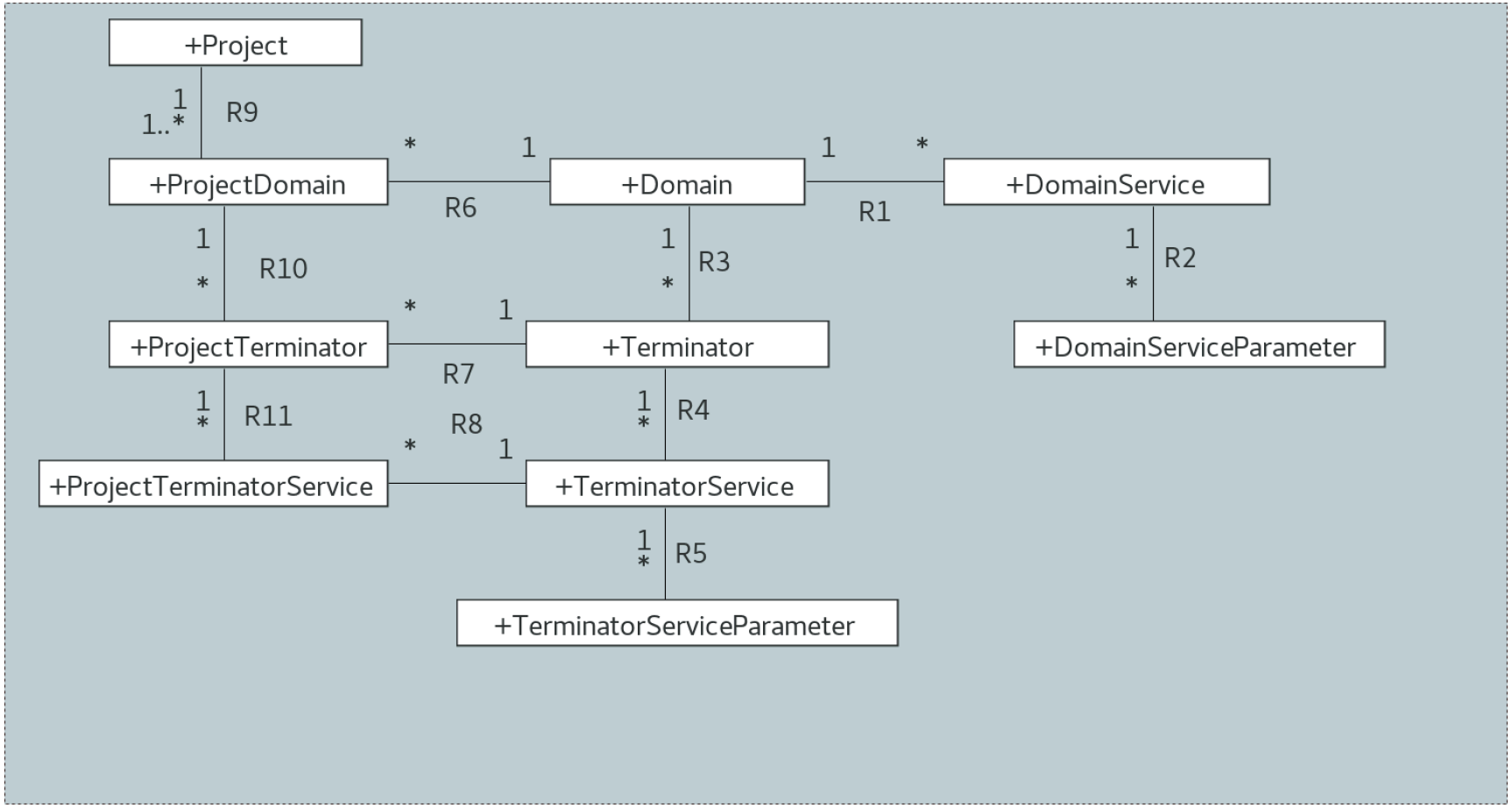
MASL Inter-domain Communications



© Crown Copyright 2019

HM Government

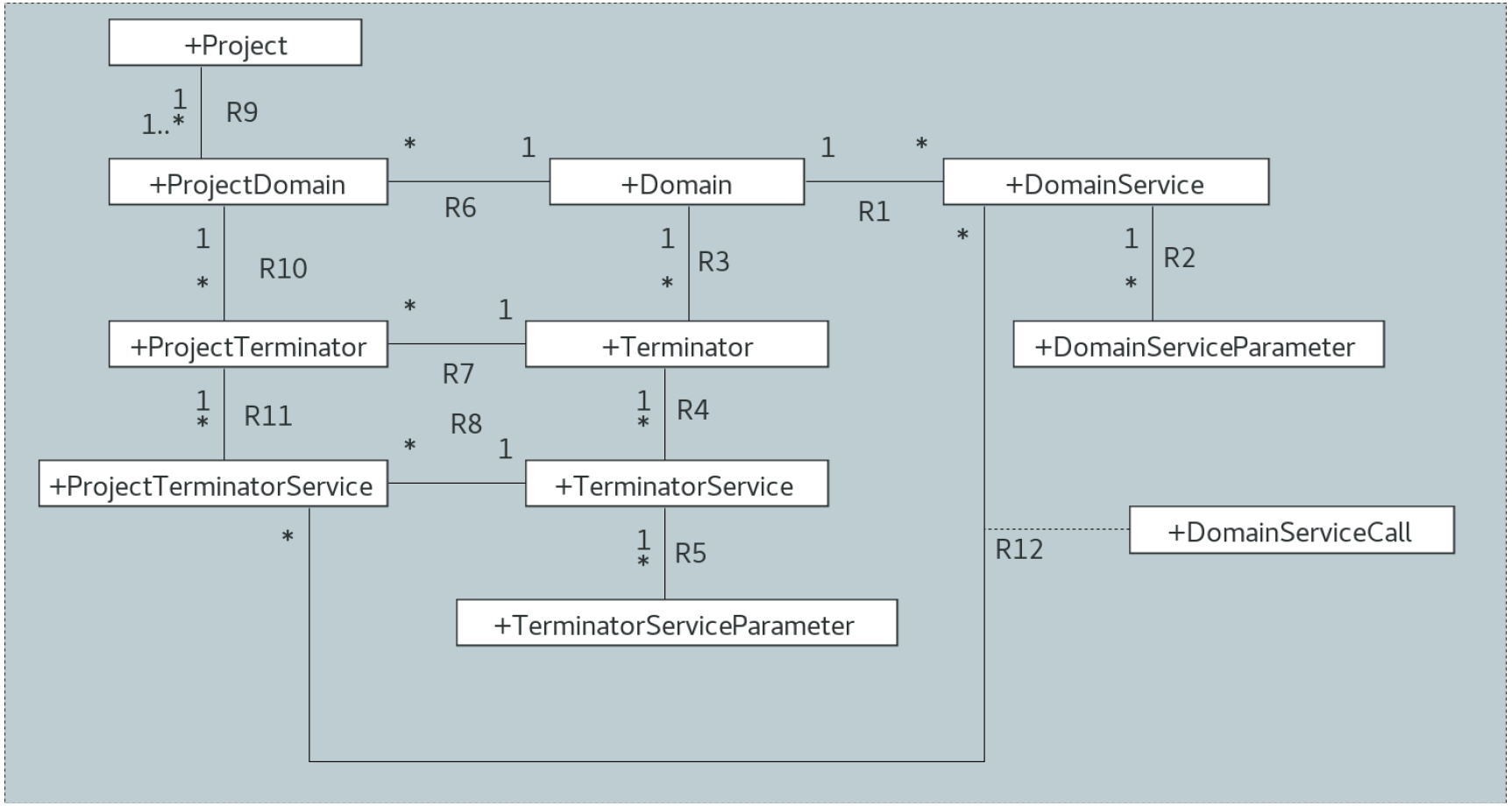
MASL Inter-domain Communications



© Crown Copyright 2019

HM Government

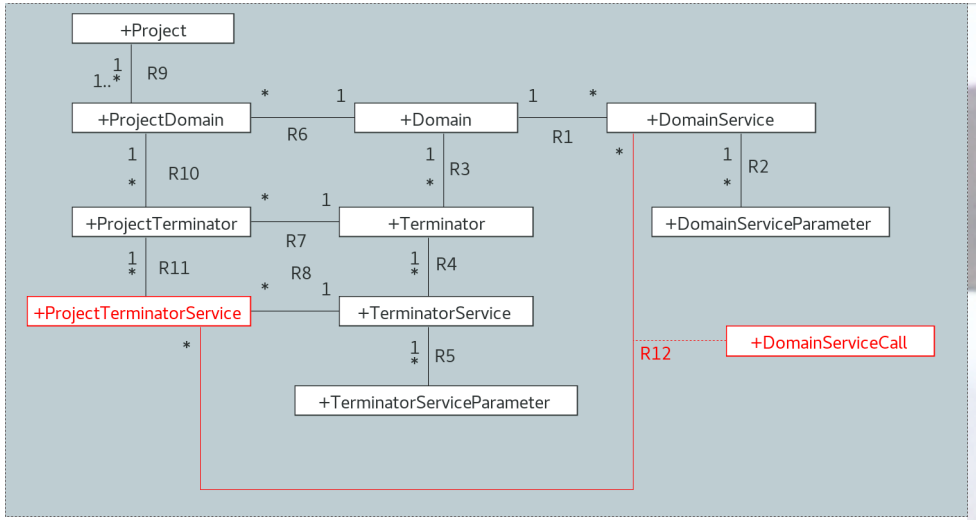
MASL Inter-domain Communications



© Crown Copyright 2019

HM Government

MASL Inter-domain Communications



- The Good
  - Full MASL action language
  - Allows complex mappings
  - Unit and type conversions
  - Many-to-many (‘forking bridge’)
  - Decoupled domains
- The Bad
  - Full MASL action language
  - Project coupled to source and destination domains.
- The Ugly
  - Decoupling using secondary interface domain and overloading messages and types. One fact in many places.

© Crown Copyright 2019

HM Government

Non-MASL Communications

- MASL Interface Domain containing types and domain service
  - Code generated interface code for C++ and Java
  - Difficult to integrate 3rd party code
  - Requires coders to know MASL to define interface
  - Same interface domain used by senders and receivers (no terminators)

```
domain VendingMachine is

  public type ItemDetail is structure
    id          : string;
    description: string;
  end structure;

  public domain service selectionMade (tx_id: in string, item : in ItemDetail );

end domain;
```


© Crown Copyright 2019

HM Government

Messaging Middleware

- ‘Message Bus’
- Reliable
- Publish and Subscribe
- Primitive routing options: local, lan, wan
- Primitive subscription options: integer message-id
- 20+ year old code
- Hard to integrate into new software
- Very hard to change…

© Crown Copyright 2019

HM Government

Where we would like to be

- Industry standard message middleware
  - AMQP, STOMP, OpenWire, etc.
  - Qpid, ActiveMQ, RabbitMQ, Kafka, Kinesis, AWS SQS, etc.
  - JSON, YAML, XML, Google Protocol Buffers, XML, Avro, ASN.1 etc.
- Interface to non-MASL processes more easily
- Well defined interfaces between domains (MASL or hand-coded)
- Well defined contracts between domains (MASL or hand-coded)
- Decoupled interfaces between domains, but One Fact in One Place!

© Crown Copyright 2019

HM Government

Disclaimer

- We are in the early stages of this…
- … so thoughts and advice more than welcome!

© Crown Copyright 2019

HM Government

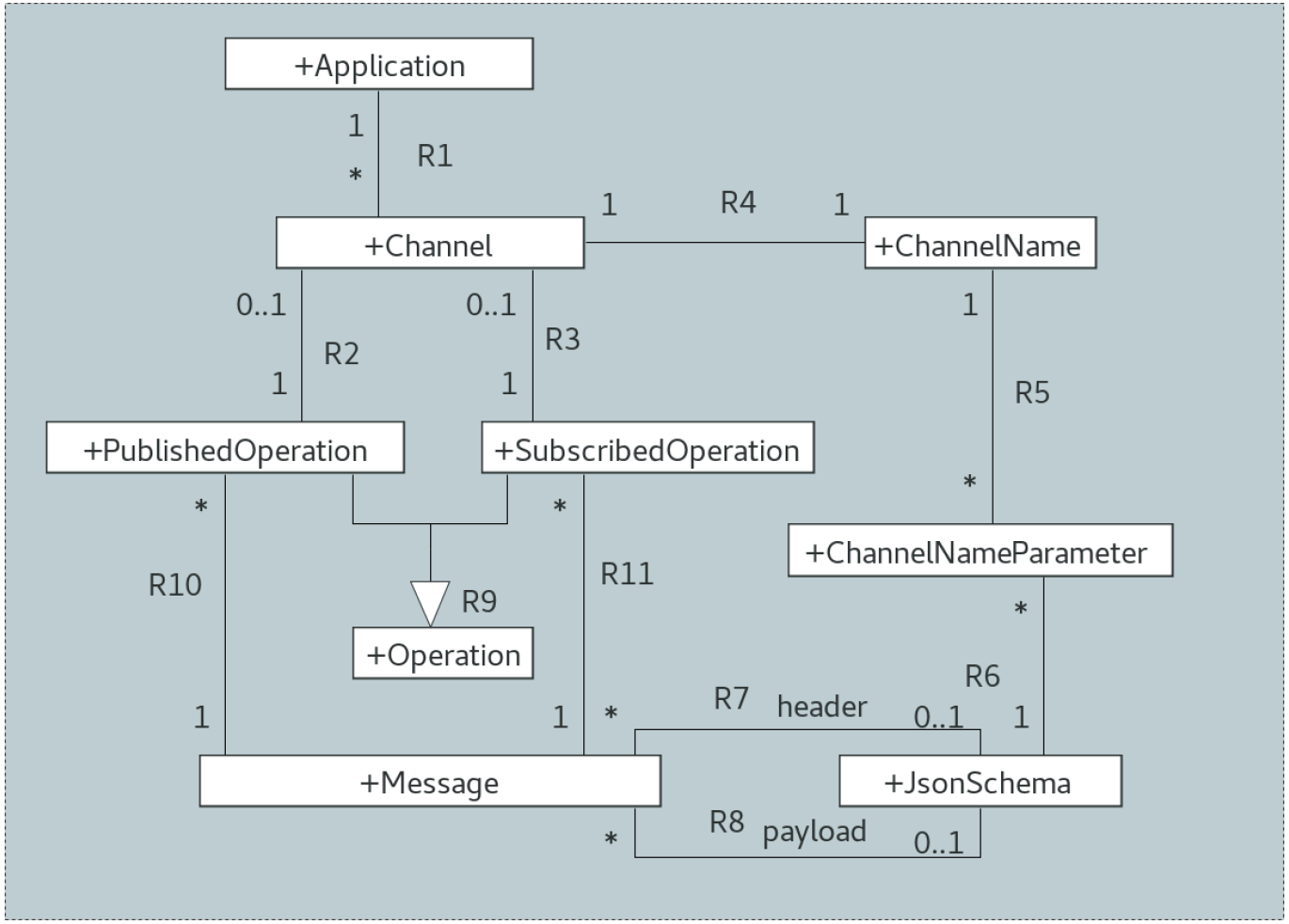
Well Defined Interfaces

- Implementation agnostic – same spec for MASL, C++, Java, etc
- Parsable – code generation, documentation generation
- Prior Art
  - JSON/XML schema – describes data, but not routing
  - OpenAPI, aka Swagger
    - Industry standard for defining REST interfaces
    - JSON Schema to define data
    - Not well-suited to publish and subscribe – could it be moulded to suit?
  - AsyncAPI
    - Based on OpenAPI, but targeted at asynchronous publish & subscribe
    - Gaining industry support

© Crown Copyright 2019

HM Government

Async API




```
asyncapi: 2.0.0
id: urn:asyncapi:vendingmachine
info:
  title: An Example Vending Machine
defaultContentType: application/json

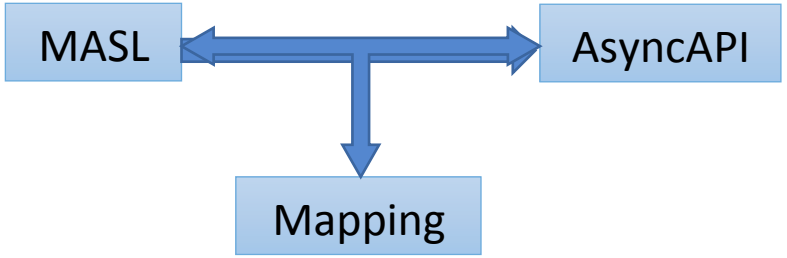
channels:
  vending/selection-made:
    subscribe:
      operationId: selection-made
        message:
          $ref: '#/components/messages/selection-made'
  vending/coin-received:
    subscribe:
      operationId: coin-received
        message:
          $ref: '#/components/messages/coin-received'
  vending/release-item:
    publish:
      operationId: release-item
        message:
          $ref: '#/components/messages/release-item'
  :
  :
components:
  messages:
    selection-made:
      header:
         $ref: ‘#/components/schemas/tx-id’
      payload:
        type: object
        properties:
          item-id:
            type: string
          description:
            type: string
    :
    :
  schemas:
    tx-id:
      type: object
      properties:
        transaction-id:
          type: string
```


© Crown Copyright 2019

HM Government

Async API

- Tools available for generating
  - API - Java Spring/JMS, Node.js
  - Documentation - Markdown, HTML
  - More on the way…

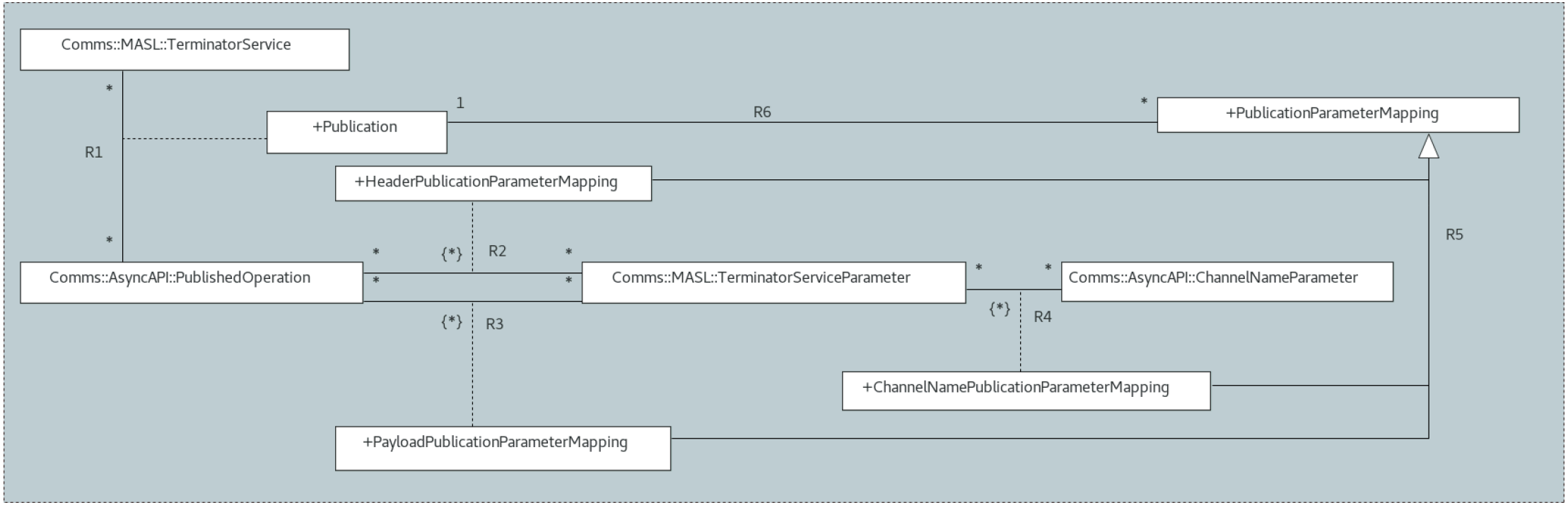


- Obviously nothing for MASL…
  - Do we generate AsyncAPI message definitions from the domain model?
  - Could we generate a skeleton domain from an AsyncAPI definition?
  - How can we map between a pre-existing domain model and a pre-existing AsyncAPI definition?

© Crown Copyright 2019

HM Government

AsyncAPI to Domain Model Mapping

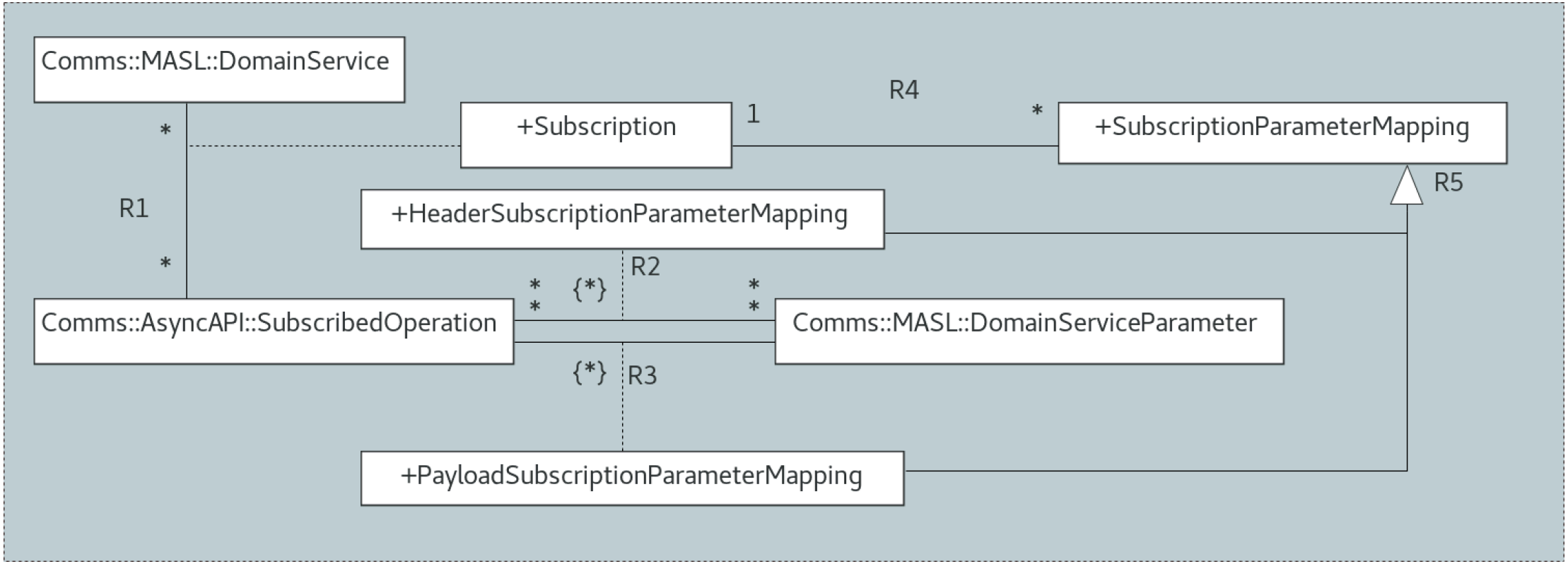


selection-made → VendingMachine::selectionMade(tx_id: in string, item : in ItemDetail)
  header::transaction-id → tx_id
  payload::item-id → item.id
  payload::description → item.description

© Crown Copyright 2019

HM Government

AsyncAPI to Domain Model Mapping



© Crown Copyright 2019

HM Government

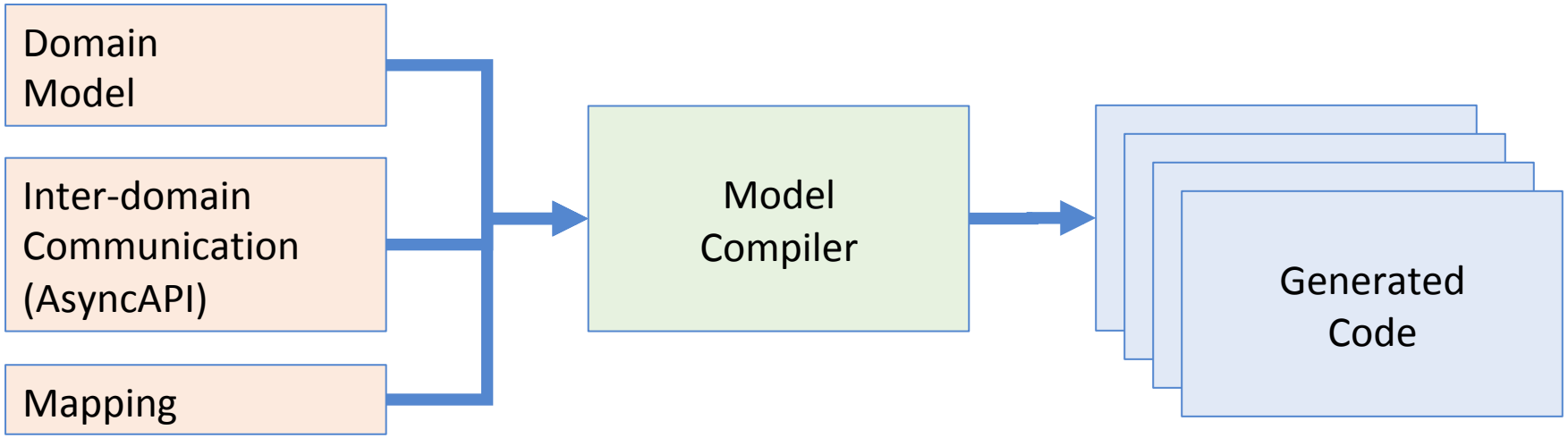
Capabilities and Limitations

- Capabilities
  - Multiple subscriptions for the same service with different parameter maps
  - Multiple publications for the same service with different parameter maps
  - Fine structure can be mapped using paths
    - box.length → cuboid.face.len
  - Parameters can be ignored
- Limitations – aka Future Extensions
  - Data transformation
    - length_mm * 25.4 → length_in
    - length * width → area
  - Constant values
    - “unspecified” → name

© Crown Copyright 2019

HM Government

Code Generation



© Crown Copyright 2019

HM Government

Abstraction

- Can/should we abstract this…?
  - Inter-domain Communication Domain (IDC)
    - Publish & Subscribe, Point-to-Point, Queue, Topic
    - Message ordering – between domains, per message type, groups
    - Delivery guarantees – at least once, no more than once, exactly once
    - Synchronous or asynchronous
    - Headers and payloads
    - Interaction Contract Specification
  - Mapping from Domain Model to IDC Model
- Tool support?

© Crown Copyright 2019

HM Government

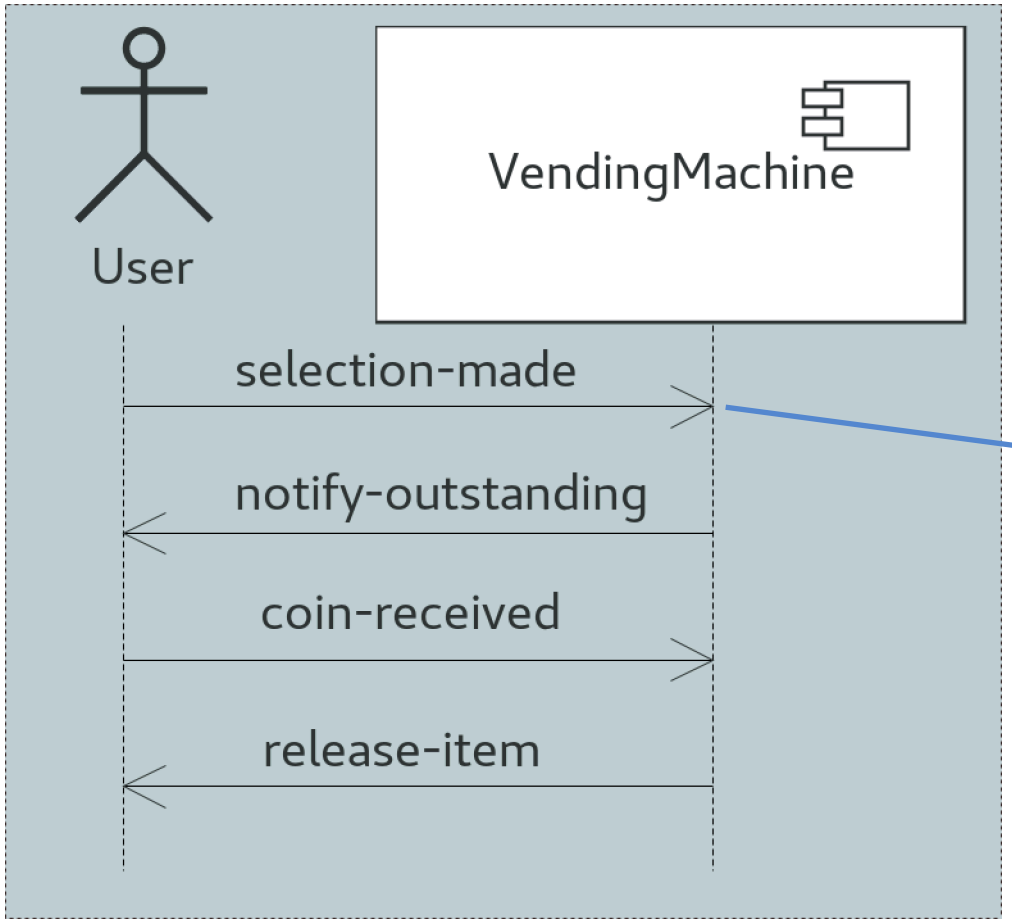
Well Defined Contracts

- Contract
  - If I send X, I expect a reply of Y or Z
  - If I receive A, I will reply with B or C
  - If I don’t receive a reply within x seconds, I will send Q
- Representation
  - Sequence Diagram
  - Collaboration Diagram
  - Activity Diagram
  - State Machine
  - Something else?

© Crown Copyright 2019

HM Government

Sequence Diagram




```
asyncapi: 2.0.0
id: urn:asyncapi:vendingmachine
info:
  title: An Example Vending Machine
defaultContentType: application/json

channels:
  vending/selection-made:
    subscribe:
      operationId: selection-made
        message:
          $ref: '#/components/messages/selection-made'
  vending/coin-received:
    subscribe:
      operationId: coin-received
        message:
          $ref: '#/components/messages/coin-received'
  vending/release-item:
    publish:
      operationId: release-item
        message:
          $ref: '#/components/messages/release-item'
  :
  :
components:
  messages:
    selection-made:
      header:
         $ref: ‘#/components/schemas/tx-id’
      payload:
        type: object
        properties:
          item-id:
            type: string
          description:
            type: string
    :
    :
  schemas:
    tx-id:
      type: object
      properties:
        transaction-id:
          type: string
```


© Crown Copyright 2019

HM Government

Sequence Diagram

```
@startuml
actor User as U
participant "Vending Machine" as VM
activate U
U ->> VM++ : [[vm.yml#/components/messages/selection-made selection-made]]
U <<- VM : [[vm.yml#/components/messages/notify-outstanding notify-outstanding]]
alt Cancel
  U ->> VM : [[vm.yml#/components/messages/cancel cancel]]
  U <<- VM : [[vm.yml#/components/messages/cancelled cancelled]]
else Normal
  loop
    U ->> VM : [[vm.yml#/components/messages/coin-received coin-received]]
    U <<- VM : [[vm.yml#/components/messages/notify-outstanding notify-outstanding]]
    break
       U <<- VM : [[vm.yml#/components/messages/release-item release-item]]
    end
    break
      U ->> VM : [[vm.yml#/components/messages/cancel cancel]]
      U <<- VM-- : [[vm.yml#/components/messages/cancelled cancelled]]
    end
  end
end
deactivate U
@enduml
```

```
asyncapi: 2.0.0
id: urn:asyncapi:vendingmachine
info:
  title: An Example Vending Machine
defaultContentType: application/json
x-sequence-diagrams:
  - role: VM
    url:  vm.plantuml
channels:
  vending/selection-made:
    subscribe:
      operationId: selection-made
        message:
          $ref: '#/components/messages/selection-made'
  vending/coin-received:
    subscribe:
      operationId: coin-received
        message:
          $ref: '#/components/messages/coin-received'
  vending/release-item:
    publish:
      operationId: release-item
        message:
          $ref: '#/components/messages/release-item'
  :
  :
components:
  messages:
    selection-made:
      header:
         $ref: ‘#/components/schemas/tx-id’
      payload:
        type: object
        properties:
          item-id:
            type: string
          description:
            type: string
    :
    :
  schemas:
    tx-id:
      type: object
      properties:
        transaction-id:
          type: string
```

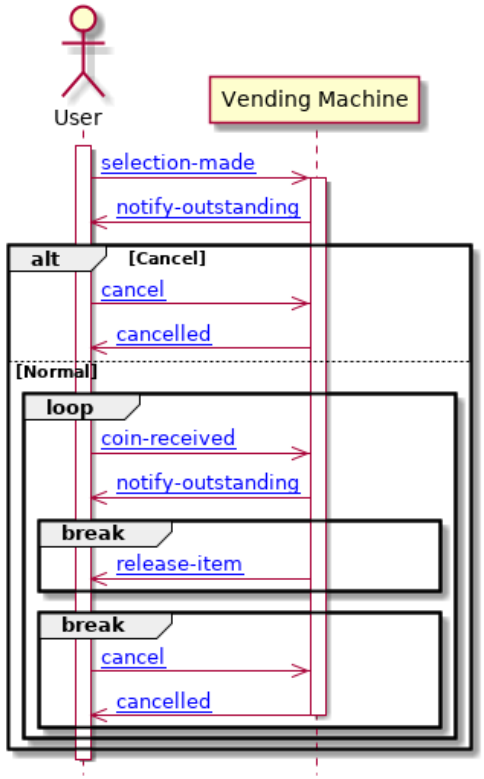

© Crown Copyright 2019

HM Government

Sequence Diagram

```
@startuml
actor User as U
participant "Vending Machine" as VM
activate U
U ->> VM++ : [[vm.yml#/components/messages/selection-made selection-made]]
U <<- VM : [[vm.yml#/components/messages/notify-outstanding notify-outstanding]]
alt Cancel
  U ->> VM : [[vm.yml#/components/messages/cancel cancel]]
  U <<- VM : [[vm.yml#/components/messages/cancelled cancelled]]
else Normal
  loop
    U ->> VM : [[vm.yml#/components/messages/coin-received coin-received]]
    U <<- VM : [[vm.yml#/components/messages/notify-outstanding notify-
outstanding]]
    break
       U <<- VM : [[vm.yml#/components/messages/release-item release-item]]
    end
    break
      U ->> VM : [[vm.yml#/components/messages/cancel cancel]]
      U <<- VM-- : [[vm.yml#/components/messages/cancelled cancelled]]
    end
  end
end
deactivate U
@enduml
```

PlantUML




```
asyncapi: 2.0.0
id: urn:asyncapi:vendingmachine
info:
  title: An Example Vending Machine
defaultContentType: application/json
x-sequence-diagrams:
  - role: VM
    url:  vm.plantuml
channels:
  vending/selection-made:
    subscribe:
      operationId: selection-made
        message:
          $ref: '#/components/messages/selection-made'
  vending/coin-received:
    subscribe:
      operationId: coin-received
        message:
          $ref: '#/components/messages/coin-received'
  vending/release-item:
    publish:
      operationId: release-item
        message:
          $ref: '#/components/messages/release-item'
  :
  :
components:
  messages:
    selection-made:
      header:
         $ref: ‘#/components/schemas/tx-id’
      payload:
        type: object
        properties:
          item-id:
            type: string
          description:
            type: string
    :
    :
  schemas:
    tx-id:
      type: object
      properties:
        transaction-id:
          type: string
```


© Crown Copyright 2019

HM Government

Well Defined Contracts

- Useful to modeller to know or specify what is expected
- Are they enforceable or testable?
- Could be checked for consistency with IDC Specification
- Documentation
- Does the contract belong to one or other domain, or neither?

© Crown Copyright 2019

HM Government

Conclusion

- Fully decoupled domains should be an aspiration
- Non-trival mappings are useful (but not too non-trivial!)
- Formal description of mapping between domains should exist outside of those domains.
- Model compiler can decide how to use that mapping
  - Doesn’t have to be asynchronous
  - Doesn’t have to be distributed
- Contracts are useful to describe requirements and expectations
  - Can validate or be validated against IDC spec
- Not just distributed systems
  - Could just as easily describe domain-domain interactions in embedded system

© Crown Copyright 2019

HM Government



© Crown Copyright 2019