Message Transformation

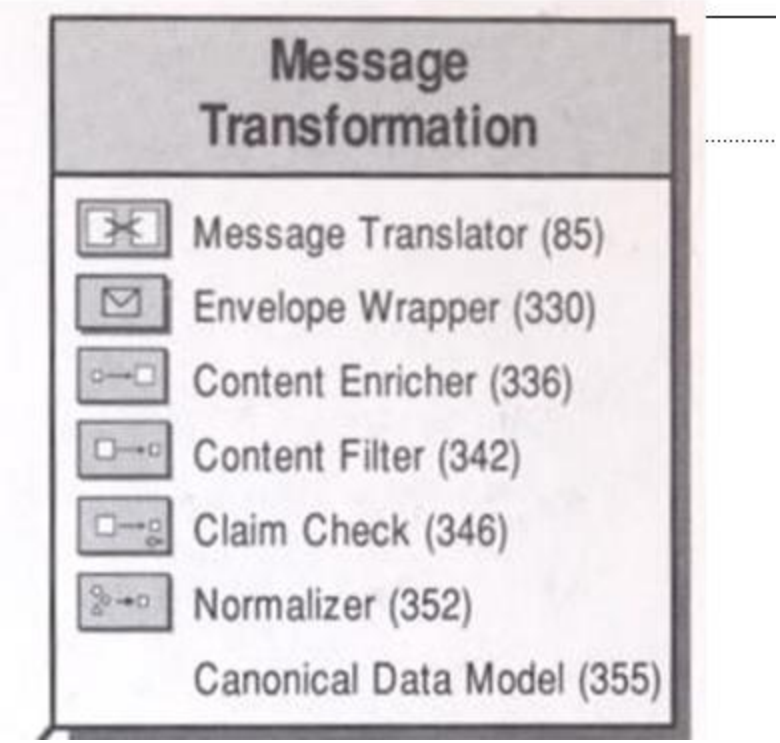
Systems Integration

PBA Softwareudvikling/BSc Software Development

Tine Marbjerg

Fall 2017

Overview of transformation patterns EIP 8



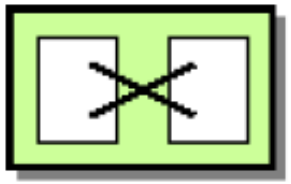
Elimination of Dependencies

Location dependency is one problem:

- *Message Channels* (60) and *Message Routers* (78) eliminate one applications awareness of another's location

Data format difference is another problem:

- *Message Translators* removes the dependency of one application being compatible with another's data format



Levels of Transformation

Systems rarely have same data format or data model, so messages must be translated in integration solution

Message Translator

Layer	Deals With	Transformation Needs (Example)	Tools/ Techniques
Data Structures (Application Layer)	Entities, associations, cardinality	Condense many-to-many relationship into aggregation.	Structural mapping patterns, custom code
Data Types	Field names, data types, value domains, constraints, code values	Convert ZIP code from numeric to string. Concatenate First Name and Last Name fields to single Name field. Replace U.S. state name with two-character code.	EAI visual transformation editors, XSL, database lookups, custom code
Data Representation	Data formats (XML, name-value pairs, fixed-length data fields, EAI vendor formats, etc.)	Parse data representation and render in a different format.	XML parsers, EAI parser/ renderer tools, custom APIs
	Character sets (ASCII, UniCode, EBCDIC)	Decrypt/encrypt as necessary.	
	Encryption/compression		
Transport	Communications protocols: TCP/IP sockets, HTTP, SOAP, JMS, TIBCO RendezVous	Move data across protocols without affecting message content.	*Channel Adapter* (127), EAI adapters

Metadata

- Transforming messages requires metadata
- Transforming messages is simplified if the *Channel Adapters* can extract metadata

RabbitMQ Java example of header + body:

```
QueueingConsumer.Delivery delivery =
        consumer.nextDelivery();
String message = new String(delivery.getBody());
delivery.getProperties().getHeaders();
```

Metadata Format

The metadata can be stored in a variety of formats, e.g. XSD's (XML Schema Definitions) for XML messages

SOAP WSDL EXAMPLE:

RPC/encoded WSDL for MyMethod

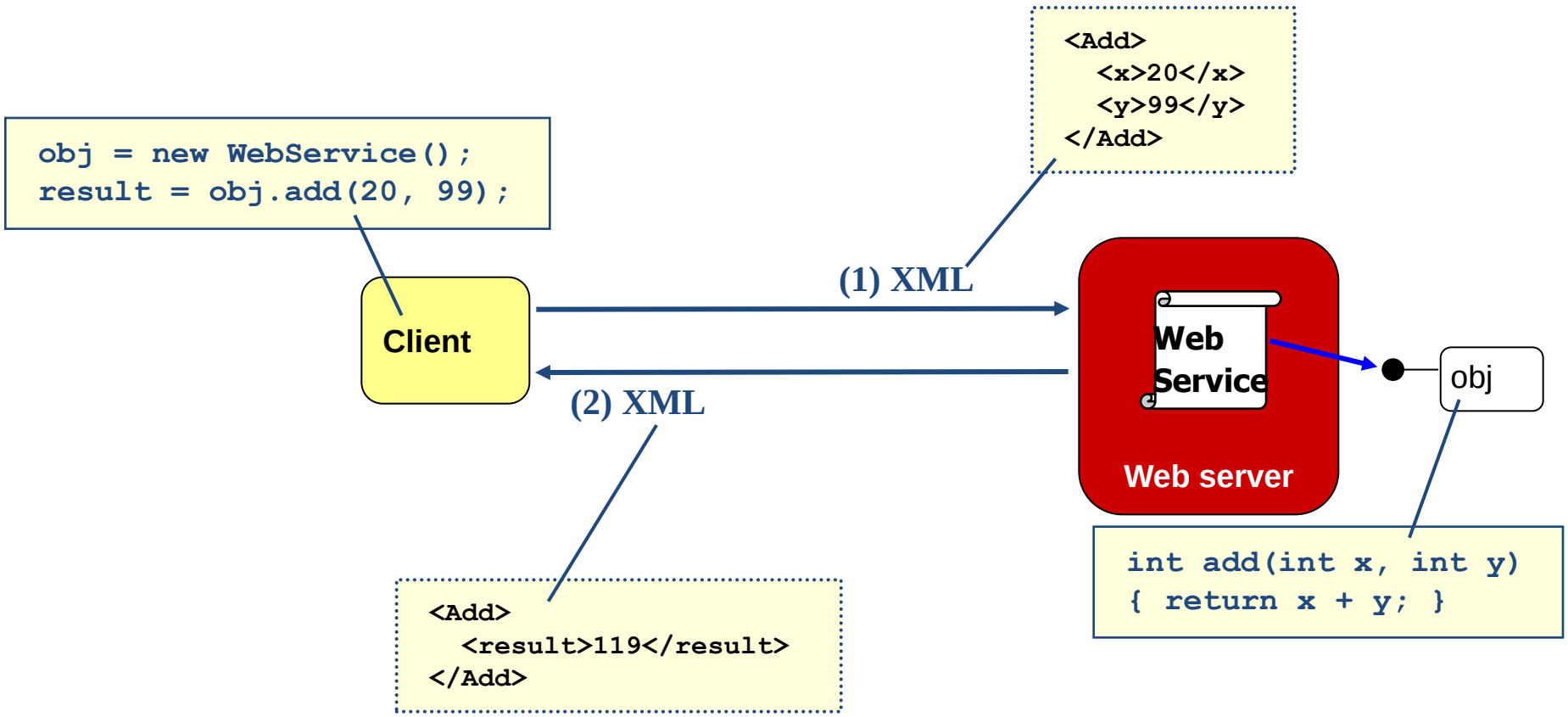
```
<message name="myMethodRequest">
    <part name="x" type="xsd:int"/>
    <part name="y" type="xsd:float"/>
</message>
<message name="empty"/>

<portType name="PT">
    <operation name="myMethod">
        <input message="myMethodRequest"/>
        <output message="empty"/>
    </operation>
</portType>
```

RPC/encoded SOAP message for myMethod

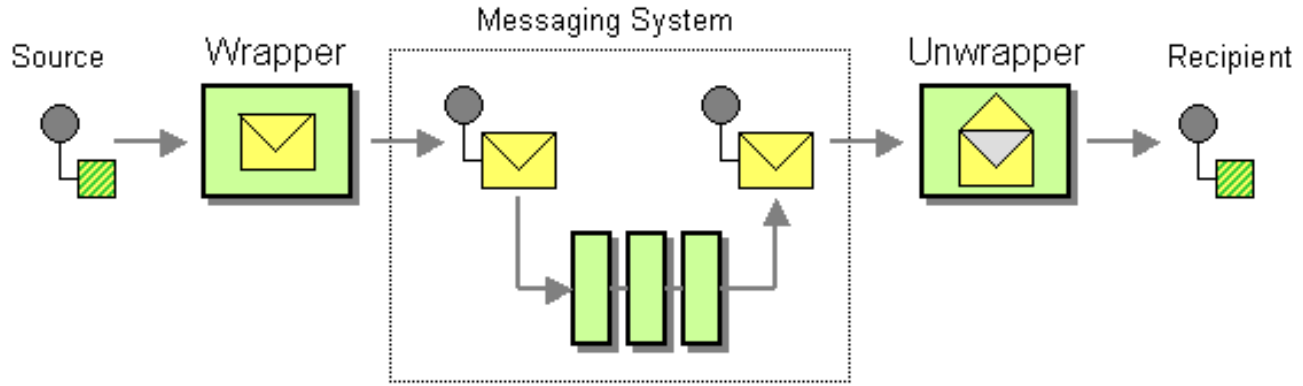
```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x xsi:type="xsd:int">5</x>
            <y xsi:type="xsd:float">5.0</y>
        </myMethod>
    </soap:body>
</soap:envelope>
```

Transformation illustrated by the SOAP Web Service Idea



Envelope Wrapper (330)

- *Envelope Wrapper* adds extra data elements to the message header which are necessary for routing, tracking, and handling the message (i.e. unique message id) in the messaging infrastructure



1. The message source publishes a message in 'raw' format
2. The wrapper transforms this message into a message format compliant with the messaging infrastructure
3. The messaging system transports the message
4. A resulting message is delivered to the "unwrapper" that reverses any modifications (e.g. removing header fields or decrypting the message)
5. The message recipient receives a 'clear text' message

Message Structure

Most messaging systems divide the message data into a header and a body

Header

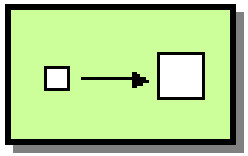
- contains fields that are used by the messaging infrastructure to manage the flow of messages

Body

- contains data to be used by the receiving application

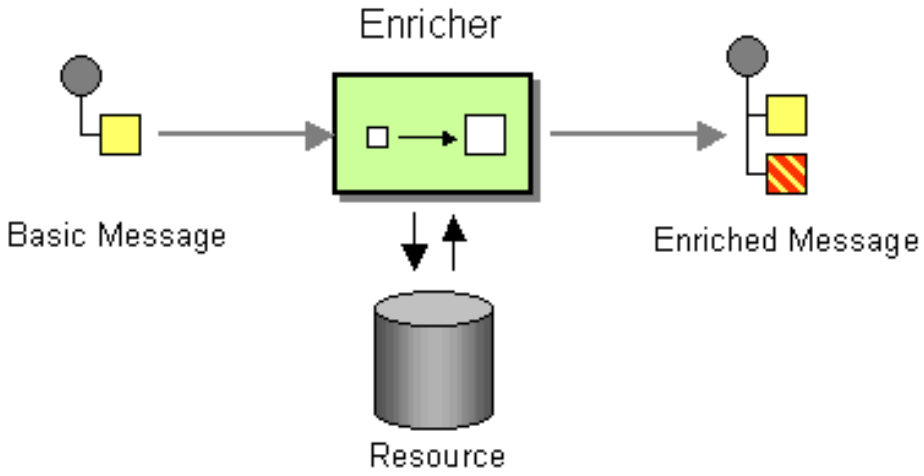
Like the Postal System!

Content Enricher(336)

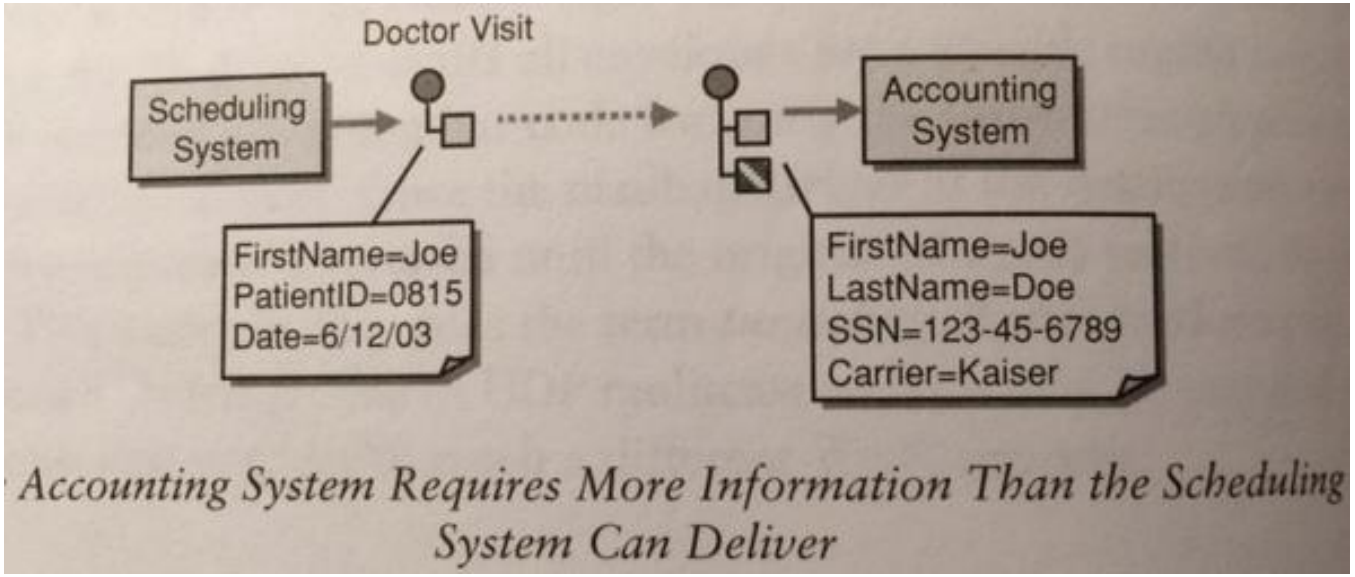


- It is common for the target system to require more information than the source system can provide.

- If the message originator does not have all the required data items available, we can use a specialized transformer, a *Content Enricher* to augment a message with missing information



Example of missing data



Accounting System Requires More Information Than the Scheduling System Can Deliver

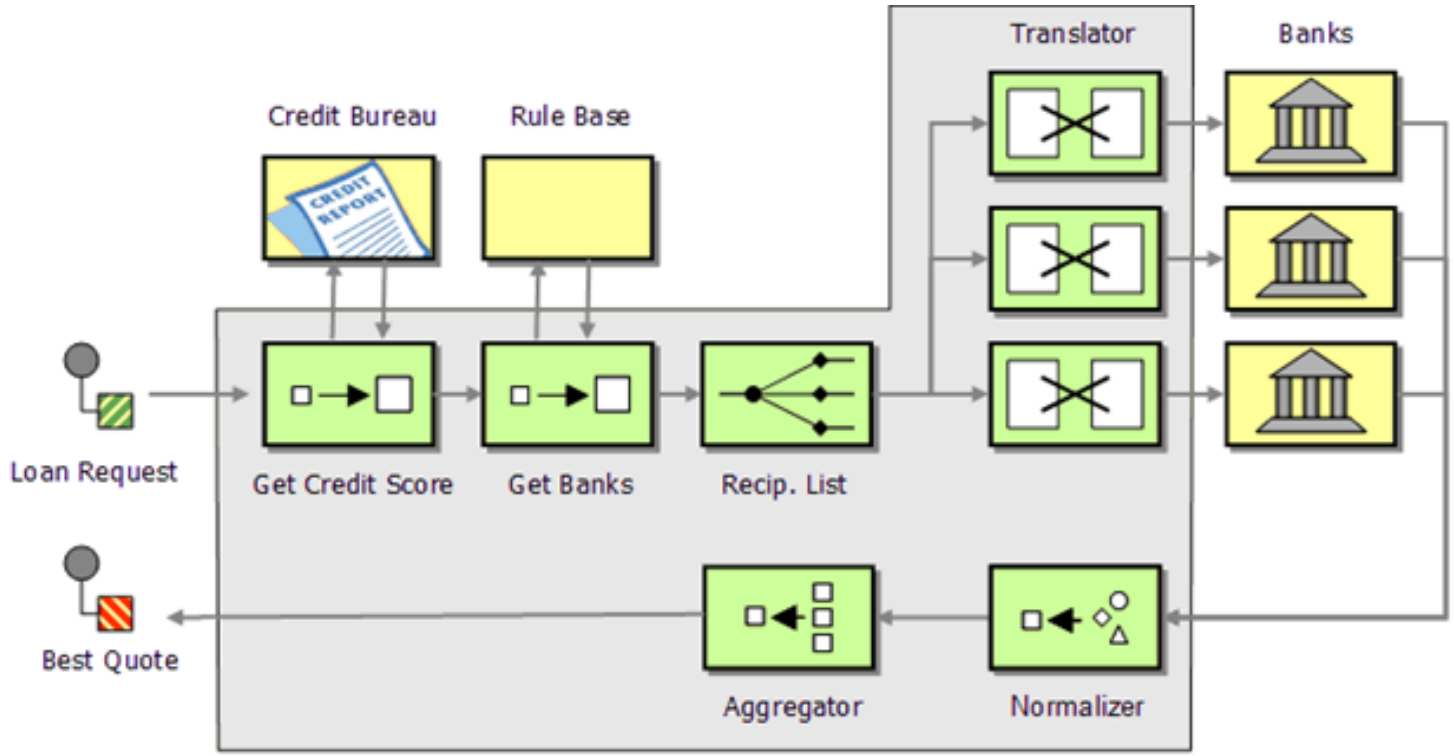
Source: EIP p. 336

Sources for the new data

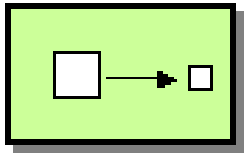
- **Computation.** Make computation on the message data
- **Environment.** Retrieve the data from the operating environment (e.g. timestamp)
- **Another system.** Retrieves the missing data from another system. Most common one.

Loan Broker Project

- Will you need content enriching?

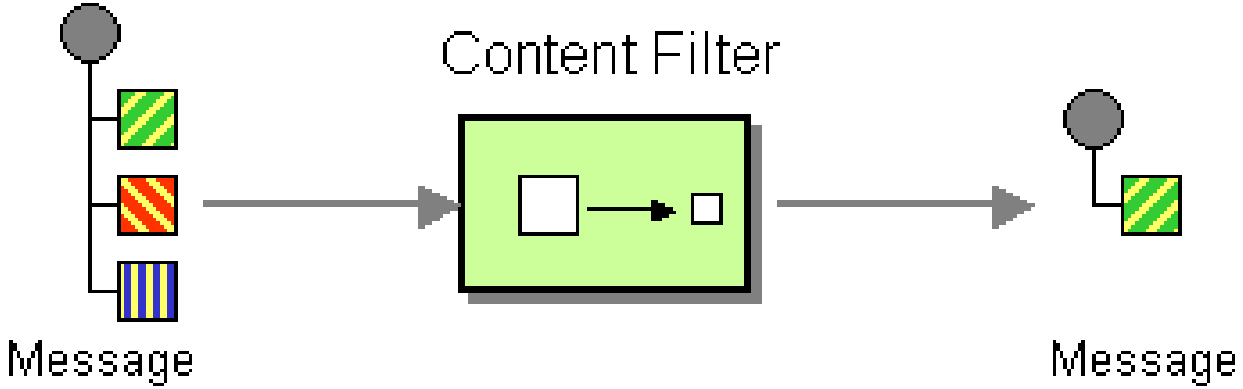


Content Filter (342)



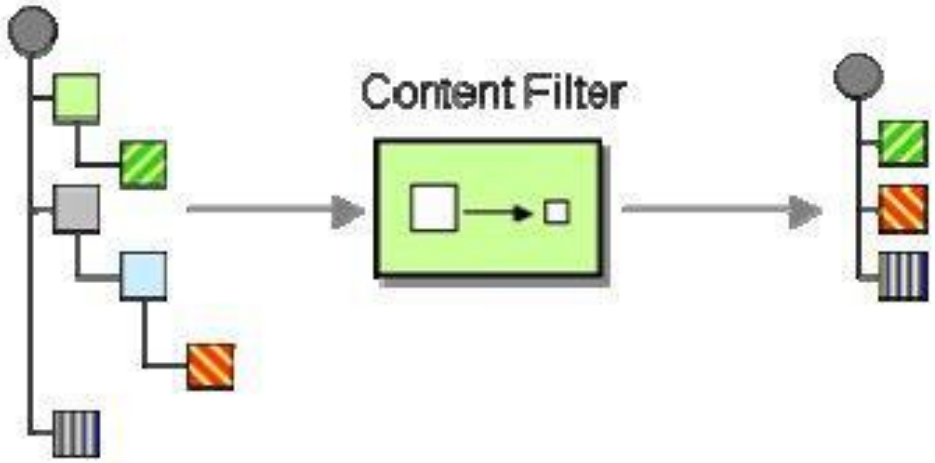
- *How do you simplify dealing with a large message, when you are interested only in a few data items?*

- Use a *Content Filter* to remove unimportant data items from a message leaving only important items

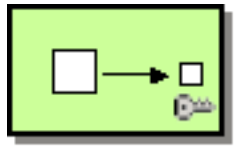


Content Filter variations

- Does not necessarily just remove data elements
- Useful to simplify the structure of the message
  - tree structures
  - levels of nested, repeating groups (e.g. normalized db structure)

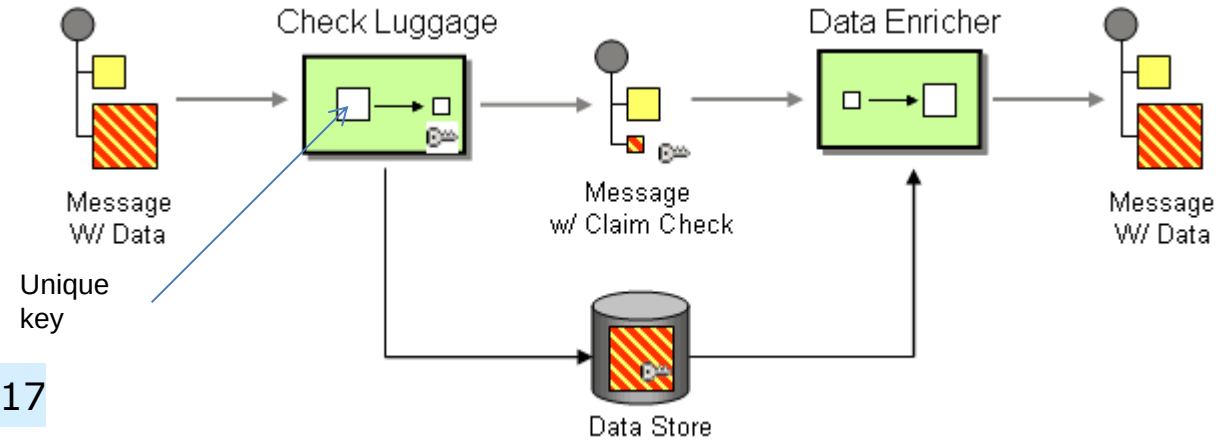


Claim Check (346)



- *How can we reduce the data volume of a message sent across the system without sacrificing information content?*

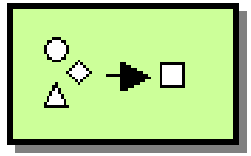
- Store message data in a persistent store and pass a *Claim Check* to subsequent components. These components can use the Claim Check to retrieve the stored information



Claim Check Steps

1. A message with data arrives
2. The Check Luggage component generates a unique key for the information
   - ***How should we choose key?***
3. The Check Luggage component extracts the data from the message and stores it in a persistent store associated with the key
   - ***How long should data be stored?***
4. It removes the persisted data from the message and adds the Claim Check
5. Another component can use a *Content Enricher(336)* to retrieve the data based on the *Claim Check*

Normalizer (352)

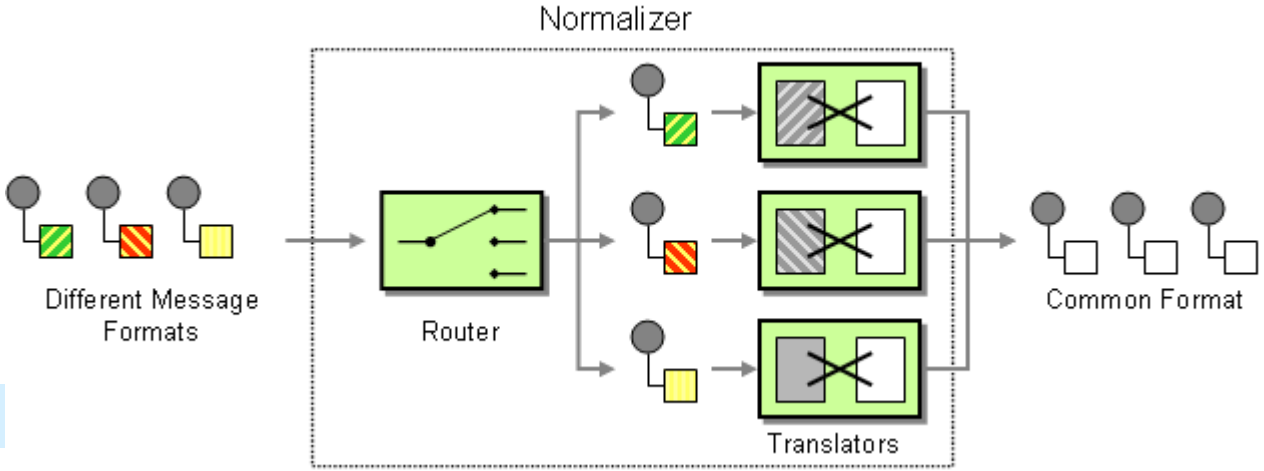


- *How do you process messages that are semantically equivalent, but arrive in a different format?*

Data Format Examples
- EDI fact
- Comma-separated files
- XML document
- Excel spreadsheet

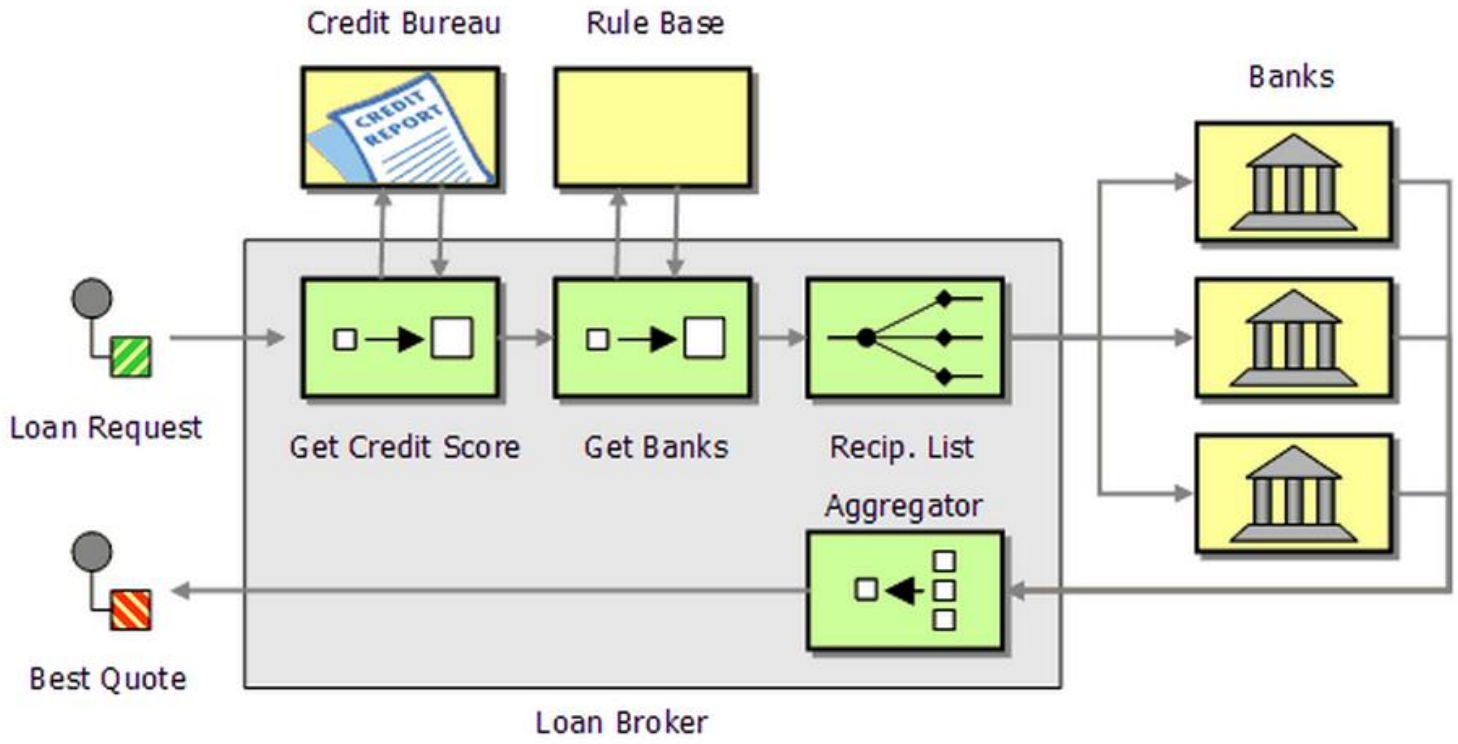
- Use a *Normalizer* to route each message type through a custom *Message Translator* so that the resulting messages match a common format

The router must know the type of incoming message, maybe by a type specifier field in the header



Loan Broker Project

- Will you need normalization?

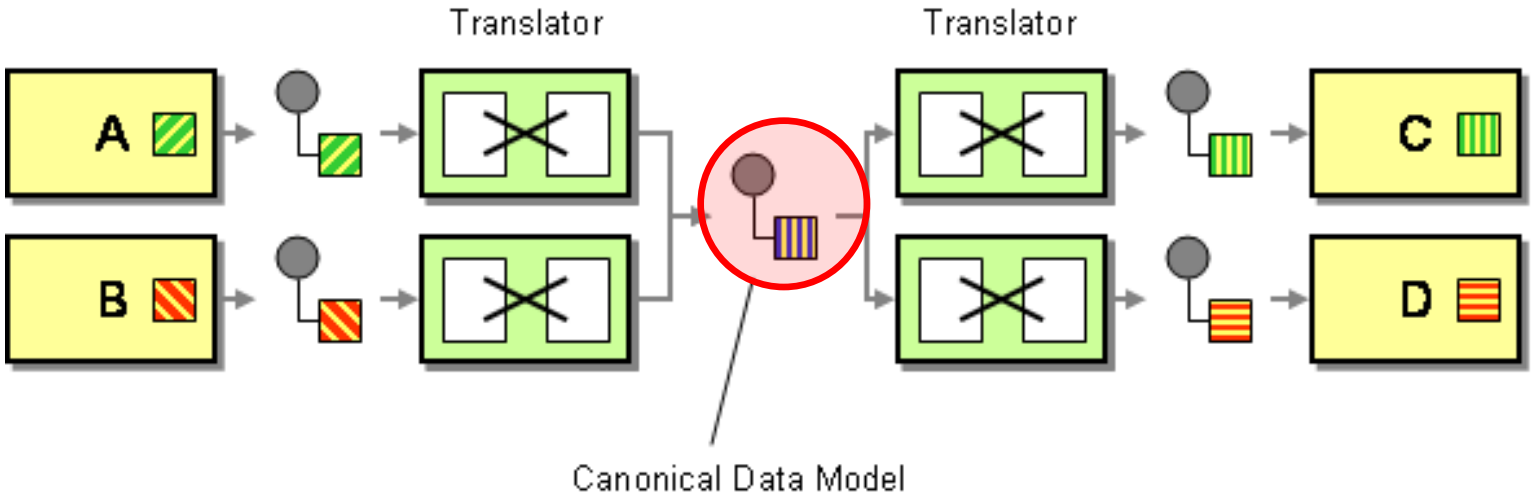


Simple Loan Broker Design

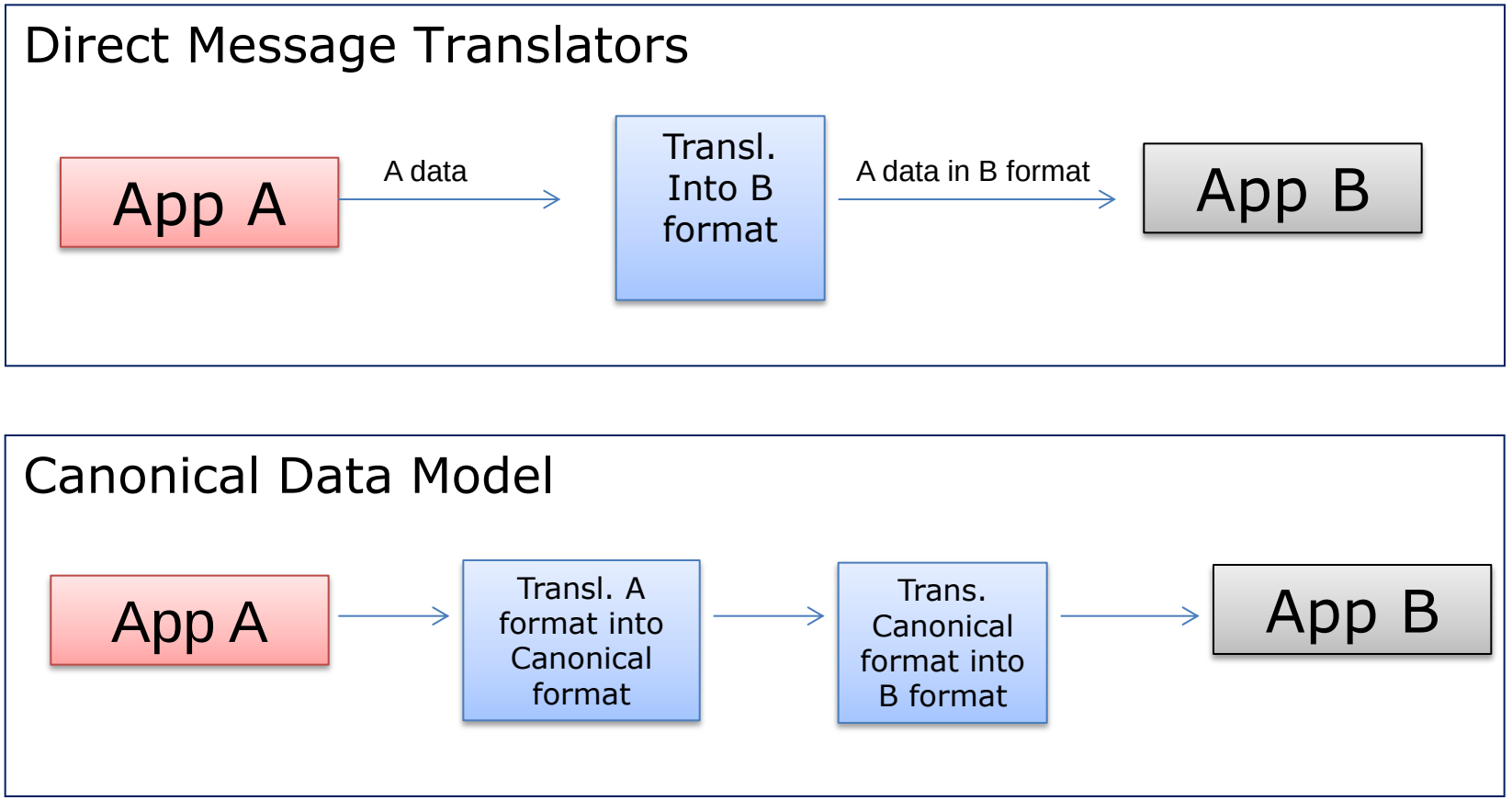
Canonical Data Model (355)

- *How can you minimize dependencies when integrating applications that use different data formats?*

- Design a *Canonical Data Model* that is independent from any specific application. Requires each application to produce and consume messages in this common format

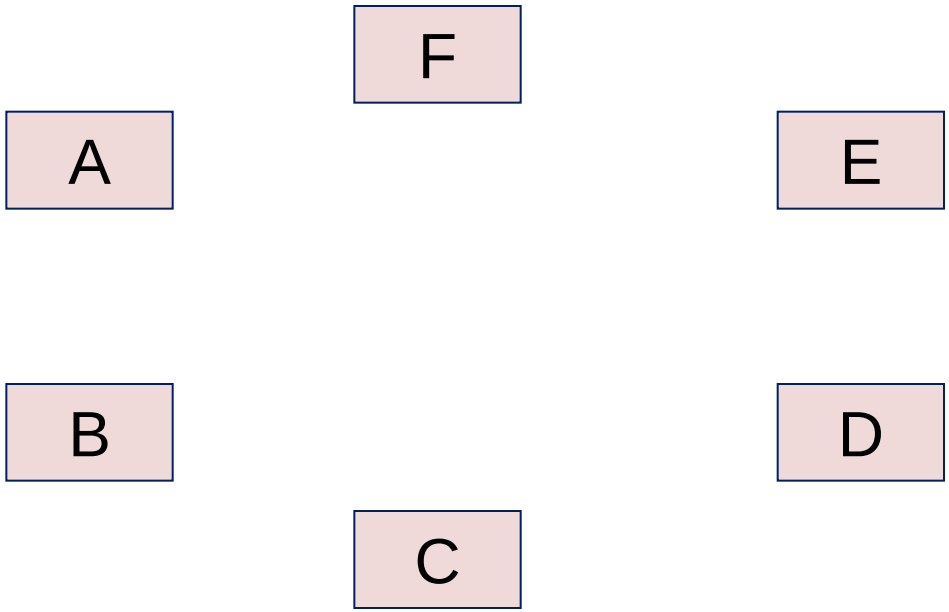


Doesn’t it Just Get More Complicated?



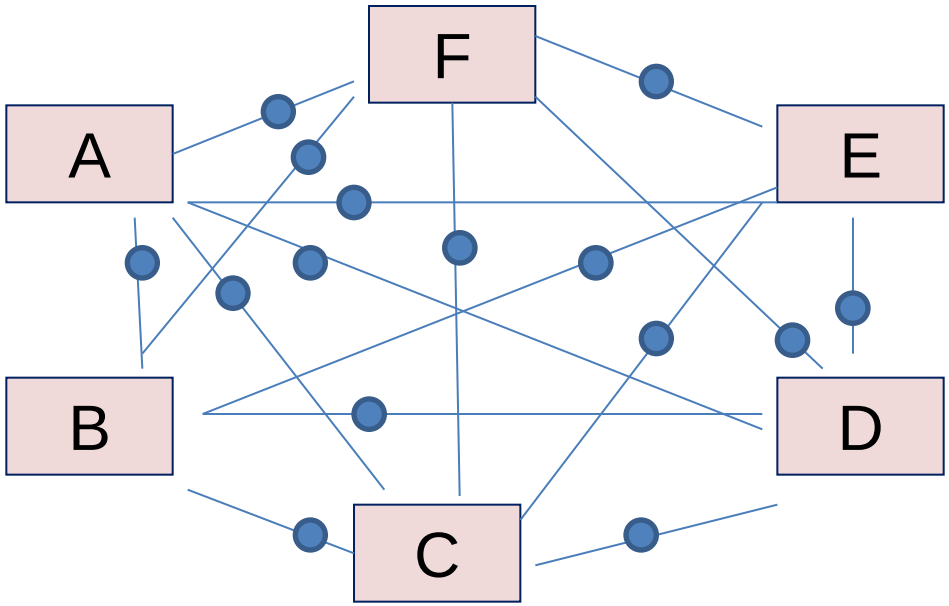
Performance vs. Maintenance

- What happens with an increasing number of systems?



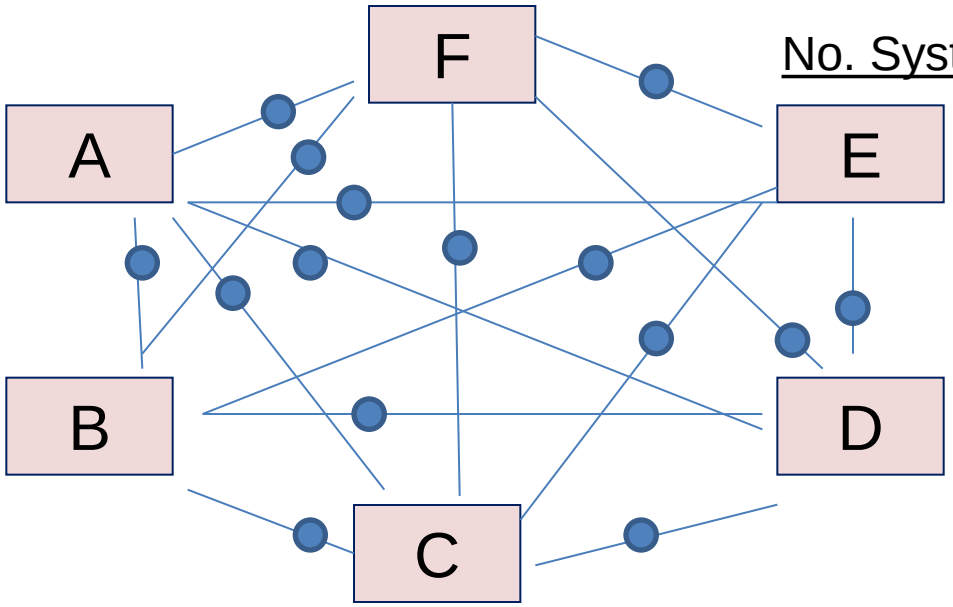
Performance vs. Maintenance

- The number of connections explode!



Performance vs. Maintenance

- Number of translations <u>without</u> Canonical Data Model*:
  n * (n – 1), i.e. O(n²)



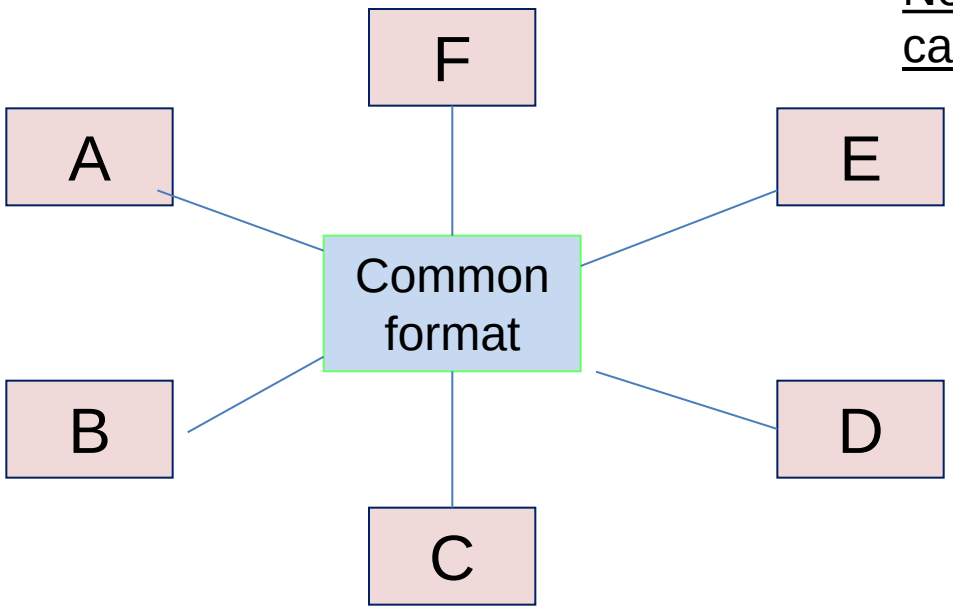
No. Systems	No. Transl (both ways)
3	6
4	12
5	20
6	30
7	42

* We assume that each application sends and receives messages to and from every other application

Performance vs. **Maintenance**

- Number of translations <u>with</u> Canonical Data Model:

  2 * n, i.e. O(n)



No. Systems	No. Transl (with canon).
3	6
4	8
5	10
6	12
7	14

- OBS! Can be difficult to design a Canonical Data Model (=enterprise data model)