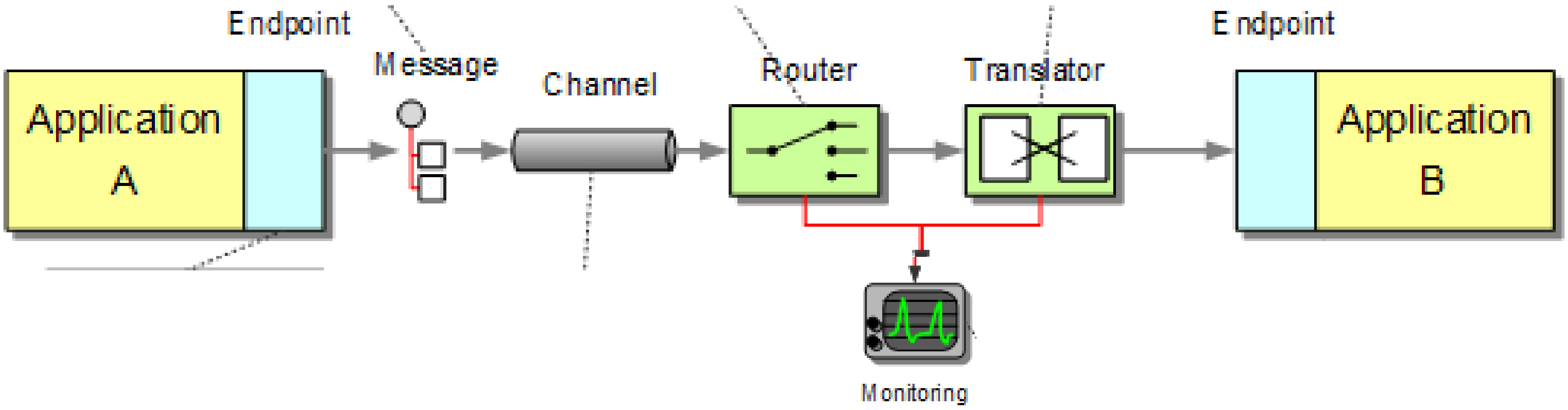
Messaging Systems

Systems Integration
PBA Softwareudvikling/BSc Software Development
Tine Marbjerg
Fall 2017

Basic Messaging Architecture – *Expressed by Symbols*

- Book uses visual language to describe integration solutions by means of patterns



Other Integration Styles?

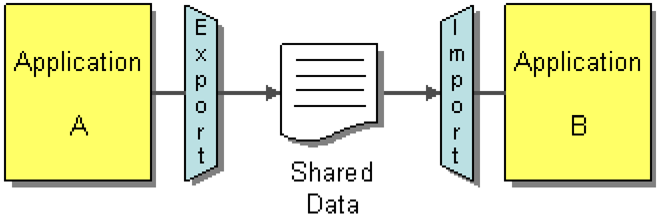
Application
A
?
Application
B

Application Integration Styles (EIP chapter 2)

- 4 options
  - File Transfer (43)
  - Shared Database (47)
  - Remote Procedure Invocation (50)
  - Messaging (53)

- Pattern order above reflects an increasing order of sophistication, but also increasing complexity

File Transfer

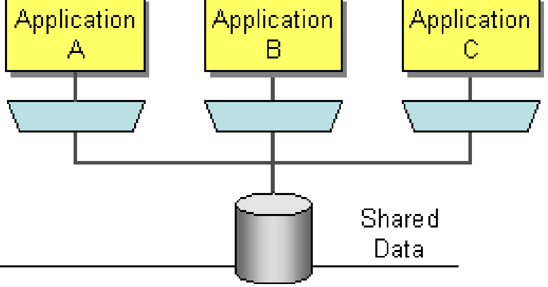


- Each application
  - produces files of shared data for others to consume,
  - consumes files that others have produced

- Data oriented

Pro	Con
Simple technology	File processing is expensive
Universal storage mechanism	Stale data due to infrequent updates (out of sync)
Decoupled from applications	No data format enforcement

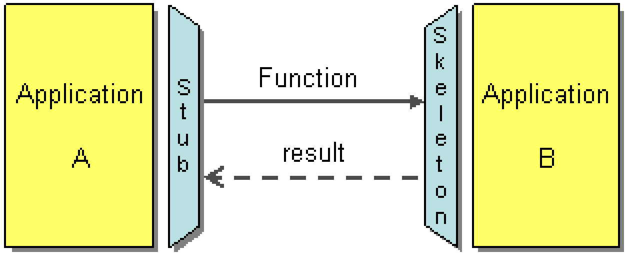
Shared Database



- Applications store the data they wish to share in a common database
- Data oriented

Pro	Con
Data available more quickly	Performance bottleneck (many read/update)
Transaction management	Deadlock
Enforce data format	Tight coupling to db (unencapsulated data structure)
	Unified schema is difficult to design

Remote Procedure Invocation

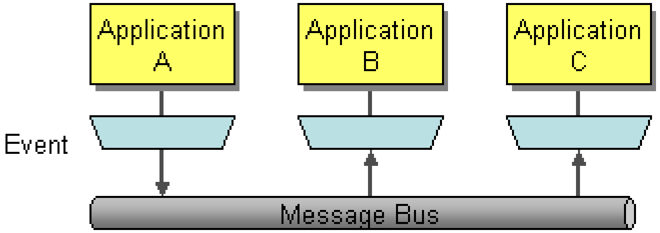


- Applications
  - expose some of their procedures to be invoked remotely,
  - invoke those exposed procedures to run behavior and exchange data

- Functionality oriented

Pro	Con
Style familiar to programmers	Remote calls are slow
Encapsulate data	Remote calls are unreliable
Can deal with semantic dissonance (multiple interfaces to same data)	Fairly tight coupling

Messaging

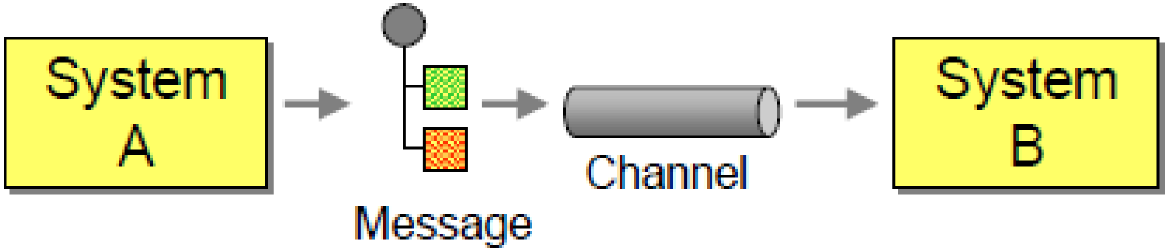


- Applications
  - connect to a common messaging system,
  - exchange data and invoke behavior using messages

- Data and functionality oriented

Pro	Con
Decoupled	A bit slower
Asynchronous	Asynchrony has higher learning curve
Reliable	Testing and debugging is harder
No data format enforcement	

Basic Messaging Architecture (EIP chapter 1)



- Channels are not part of the systems (applications)
- Channels are asynchronous & reliable
- Systems don’t know each other (loose coupling)
- Data is exchanged in self-contained messages

Basic Concepts - Overview

- Channel
  - applications transmit data through a Message Channel
- Message
  - atomic packet of data that can be transmitted on a channel
- Pipes and filters
  - Perform certain actions on the message during transmission
- Routing
  - a message passes to different channels during transmission
- Transformation
  - formatting of the message during transmission
- Endpoint
  - layer of code that works as an interface or bridge between the application and the message system
- System Management
  - Monitors flow of data, makes sure everything is up running etc.

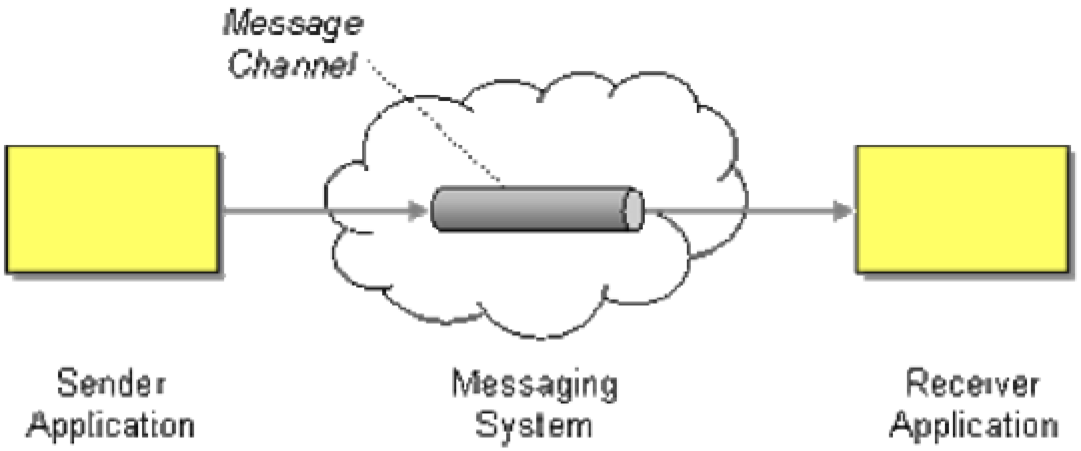
Basic Messaging Concepts (EIP chapter 3)

- Basic Concepts
  - Channel (60)
  - Message (66)
  - Pipes and Filters (70)
  - Message Router (78)
  - Message Translator (85)
  - Message Endpoint (95)

- Today’s exercises
  - Made by EIP author Gregor Hohpe
  - Give you conceptual overview of messaging
  - Called Coffee Shop exercises

Message Channel (60)

- ***How does one application communicate with another using messaging?***
- Connect the applications using a *Message Channel,* where one application writes information to the channel and the other one reads that information from the channel

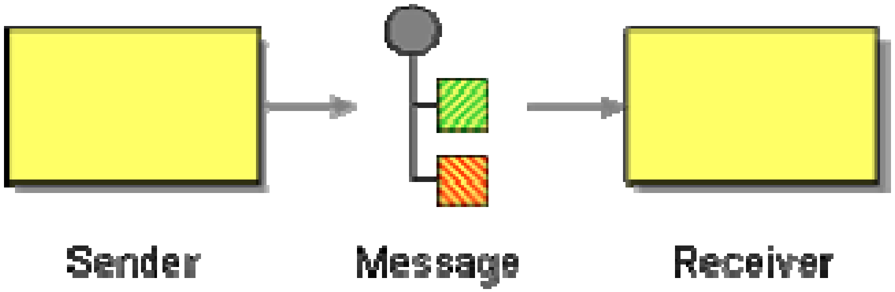


Message Channel Design

- Channels are logical addresses in the messaging system,
  - E.g. MSMQ path name syntax: **`ComputerName\QueueName`**
- Developers have to decide what channels they need for communication
- Number and purpose of channels tend to be fixed at deployment time
- A well-designed set of channels form a messaging API for a whole group of applications
- Pretty much like database design ☺

Message (66)

- ***How can two applications connected by a message channel exchange a piece of information?***
- Package the information into a *Message,* a data record that the messaging system can transmit through a message channel



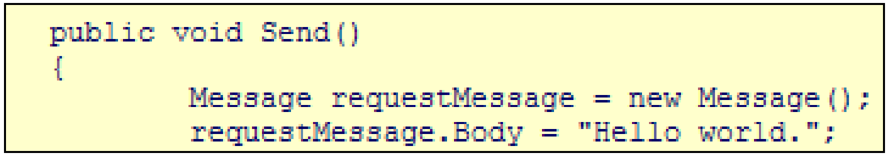
Message – Basic Parts

Header

Information used by the messaging system that describes the data being transmitted; its origin; its destination, lifetime, priority etc.

Body

The data being transmitted

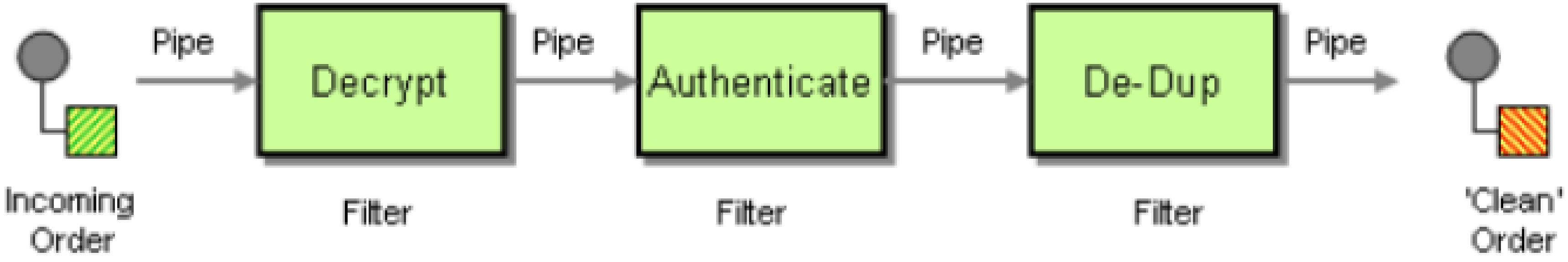


```
public void Send()
{
        Message requestMessage = new Message();
        requestMessage.Body = "Hello world.";
```

Generally ignored by the messaging system and simply transmitted as-is

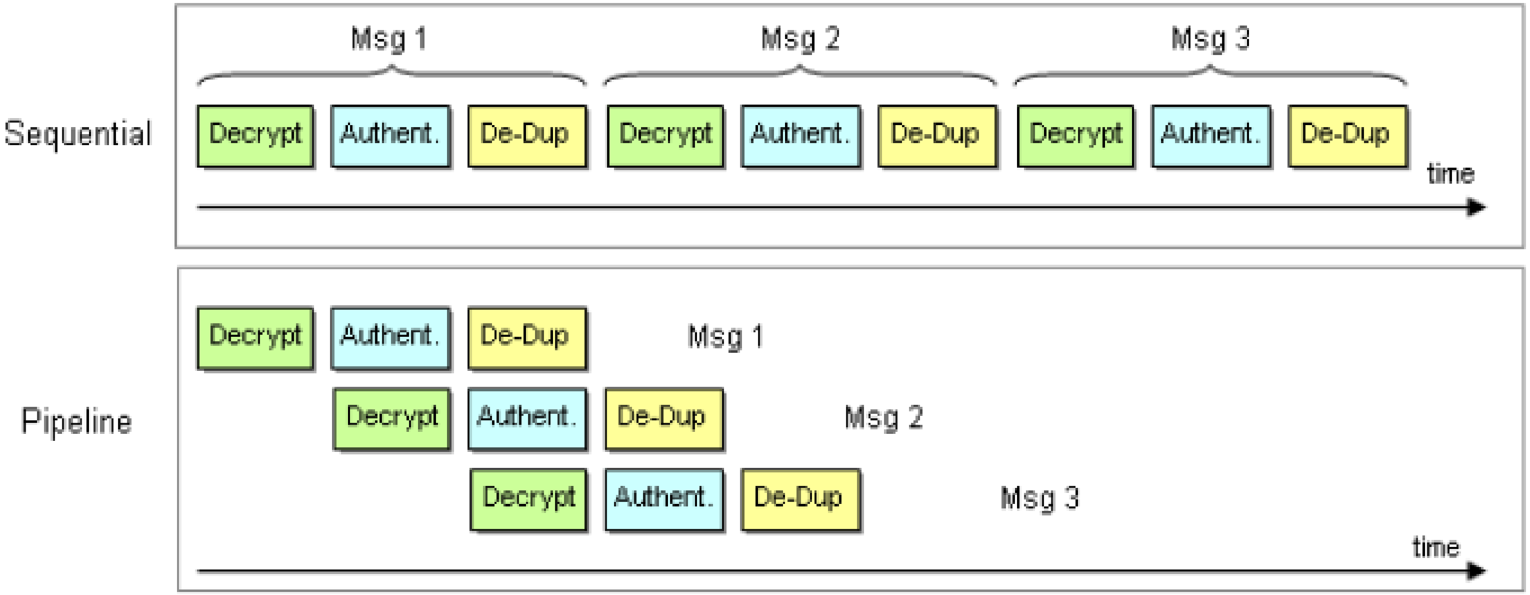
Pipes and Filters (70)

- ***How can we perform complex processing on a message while maintaining independence and flexibility?***
- Use the *Pipes and* Filters architectural style to divide a larger processing task into a sequence of smaller, independent processing steps (filters) that are connected by channels (pipes)



Pipeline Processing

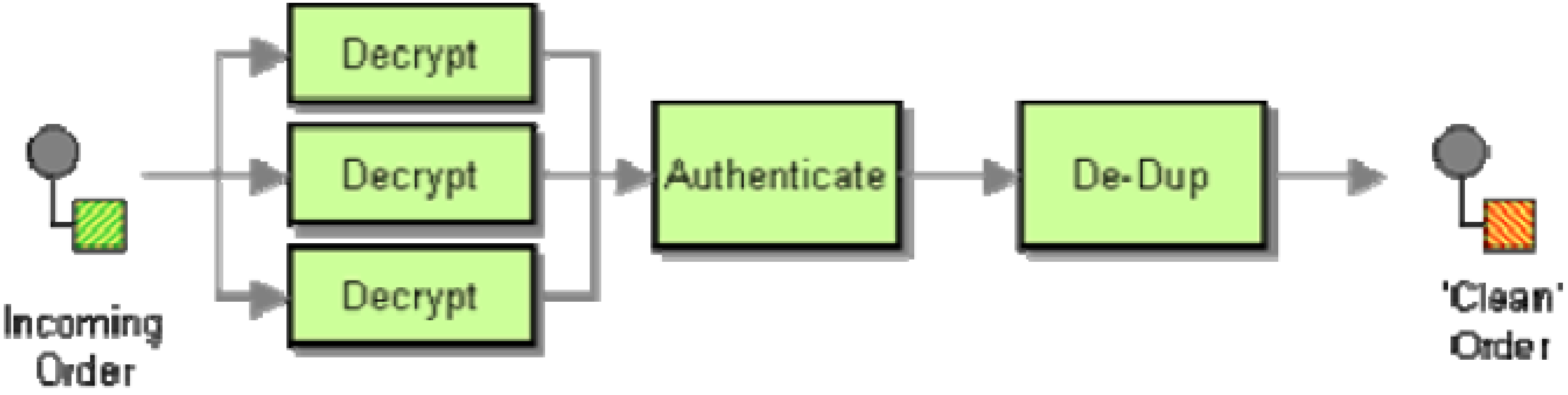
- *Pipes and Filters* allows multiple messages to be processed concurrently
- Can increase the overall system *throughput* (but not the throughput for each message)



Parallel Processing

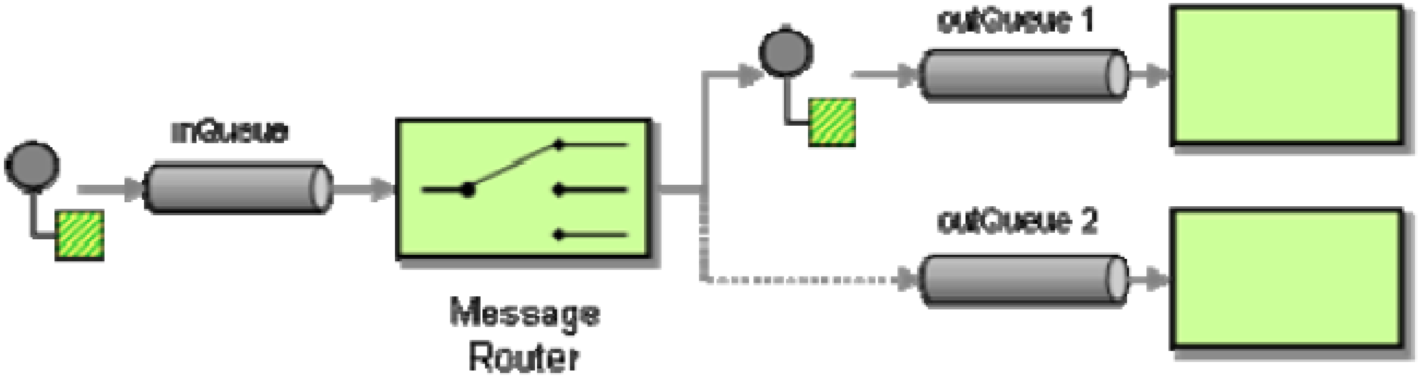
- The overall system throughput is limited by the slowest process in the chain.

- We can increase throughput with parallel processing:



Message Router (78)

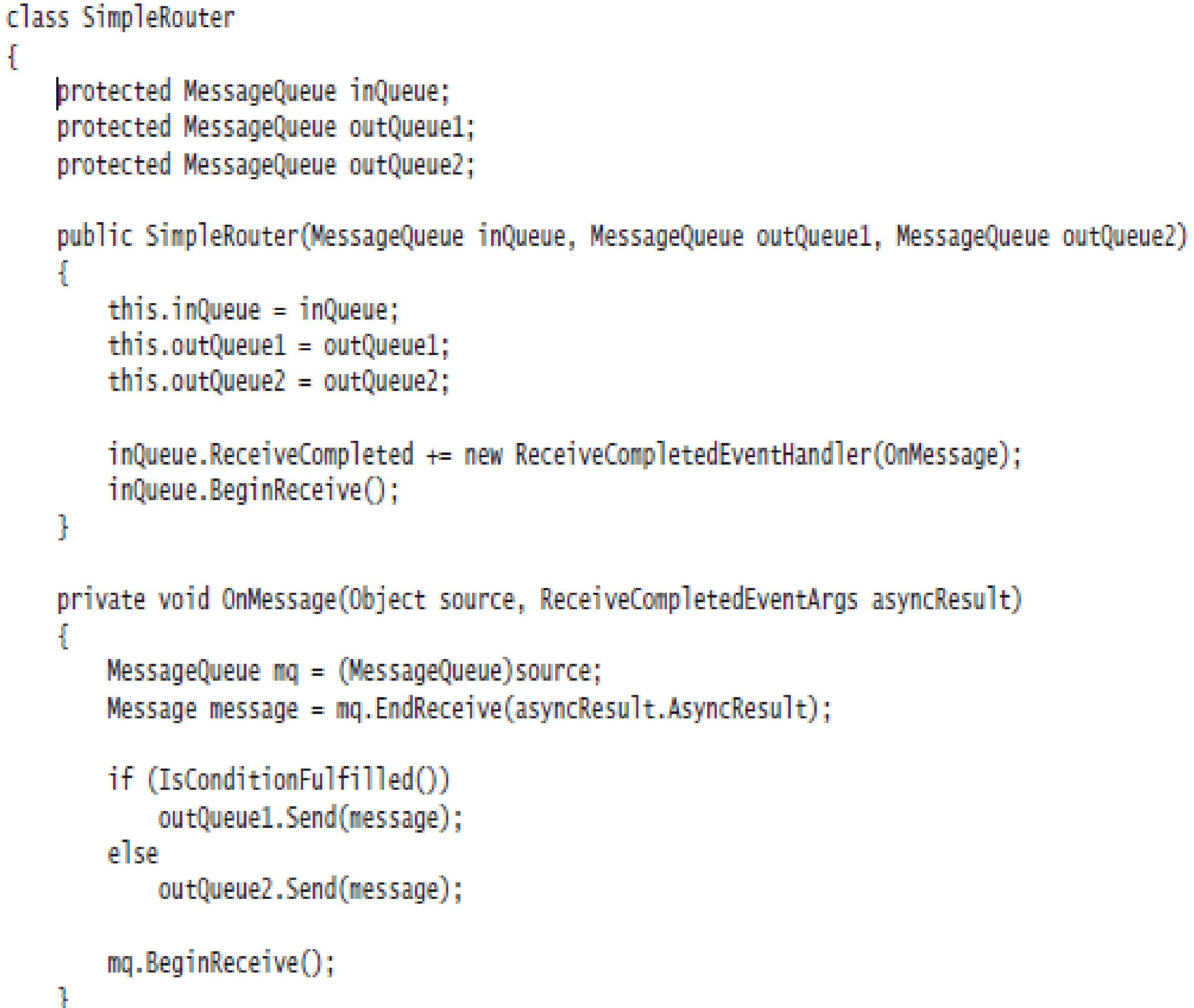
- ***How can you decouple individual processing steps so that messages can be passed to different filters depending on a set of conditions?***
- Insert a special filter, a *Message Router,* which consumes a *Message* from one *Message Channel* and republishes it to a different *Message Channel,* depending on a set of conditions



Message Router Variants

- Fixed between two channels
- Content-based
- Context-based
- Stateless or stateful

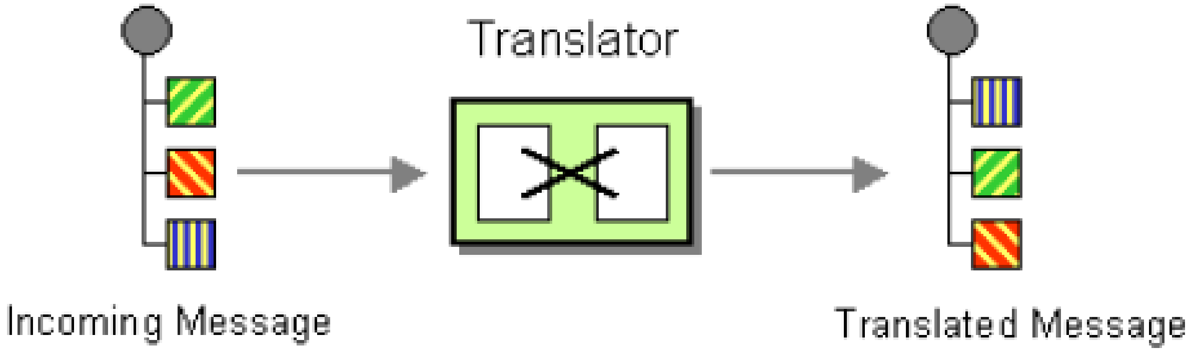
Simple Router with C# and MSMQ – (Example EIP p. 83)



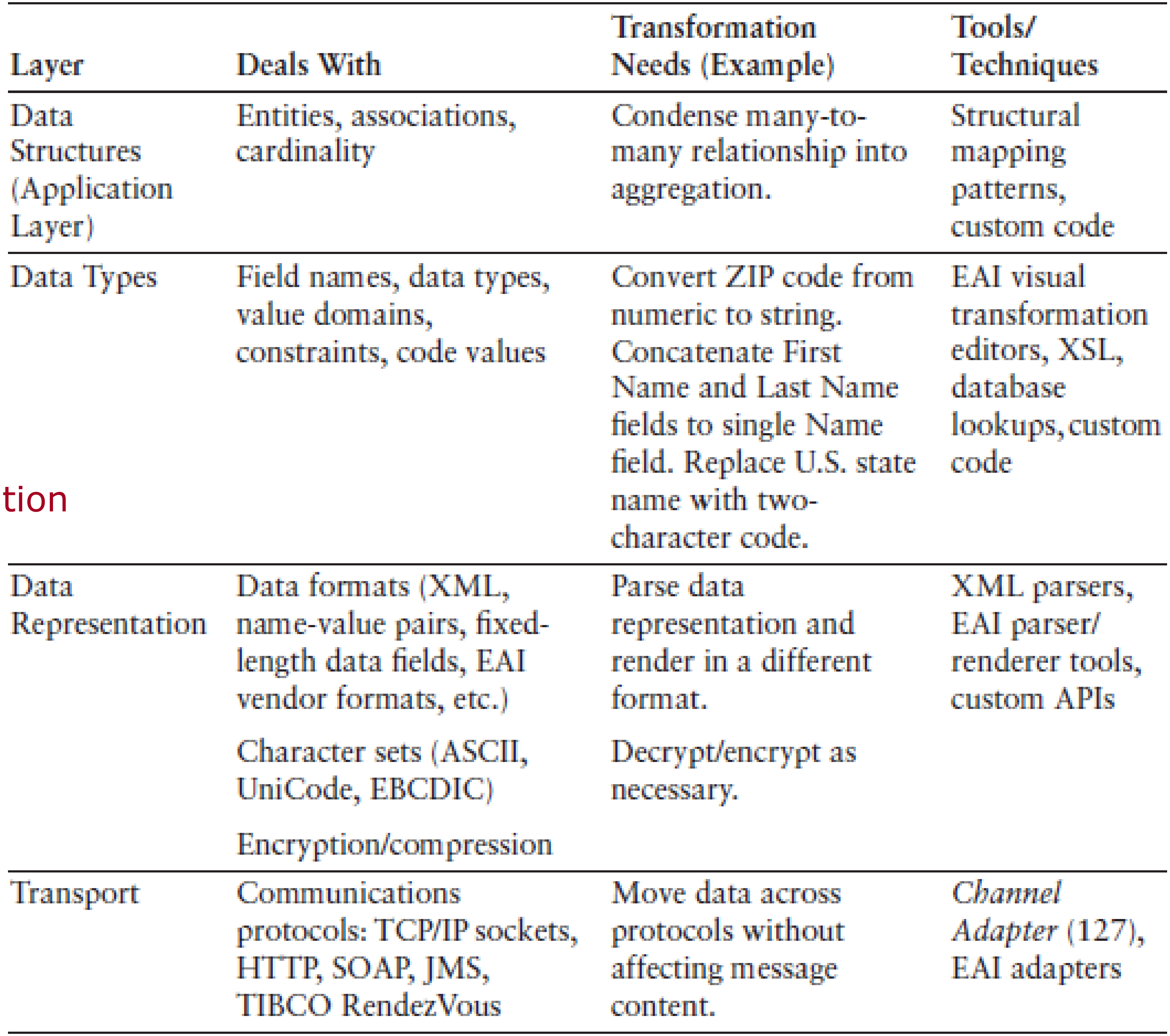
```
class SimpleRouter
{
    protected MessageQueue inQueue;
    protected MessageQueue outQueue1;
    protected MessageQueue outQueue2;

    public SimpleRouter(MessageQueue inQueue, MessageQueue outQueue1, MessageQueue outQueue2)
    {
        this.inQueue = inQueue;
        this.outQueue1 = outQueue1;
        this.outQueue2 = outQueue2;

        inQueue.ReceiveCompleted += new ReceiveCompletedEventHandler(OnMessage);
        inQueue.BeginReceive();
    }

    private void OnMessage(Object source, ReceiveCompletedEventArgs asyncResult)
    {
        MessageQueue mq = (MessageQueue)source;
        Message message = mq.EndReceive(asyncResult.AsyncResult);

        if (IsConditionFulfilled())
            outQueue1.Send(message);
        else
            outQueue2.Send(message);

        mq.BeginReceive();
    }
```

Message Translator (85)

- ***How can systems using different data formats communicate with each other using messaging?***
- Use a special filter, a *Message Translator,* to translate one data format into another



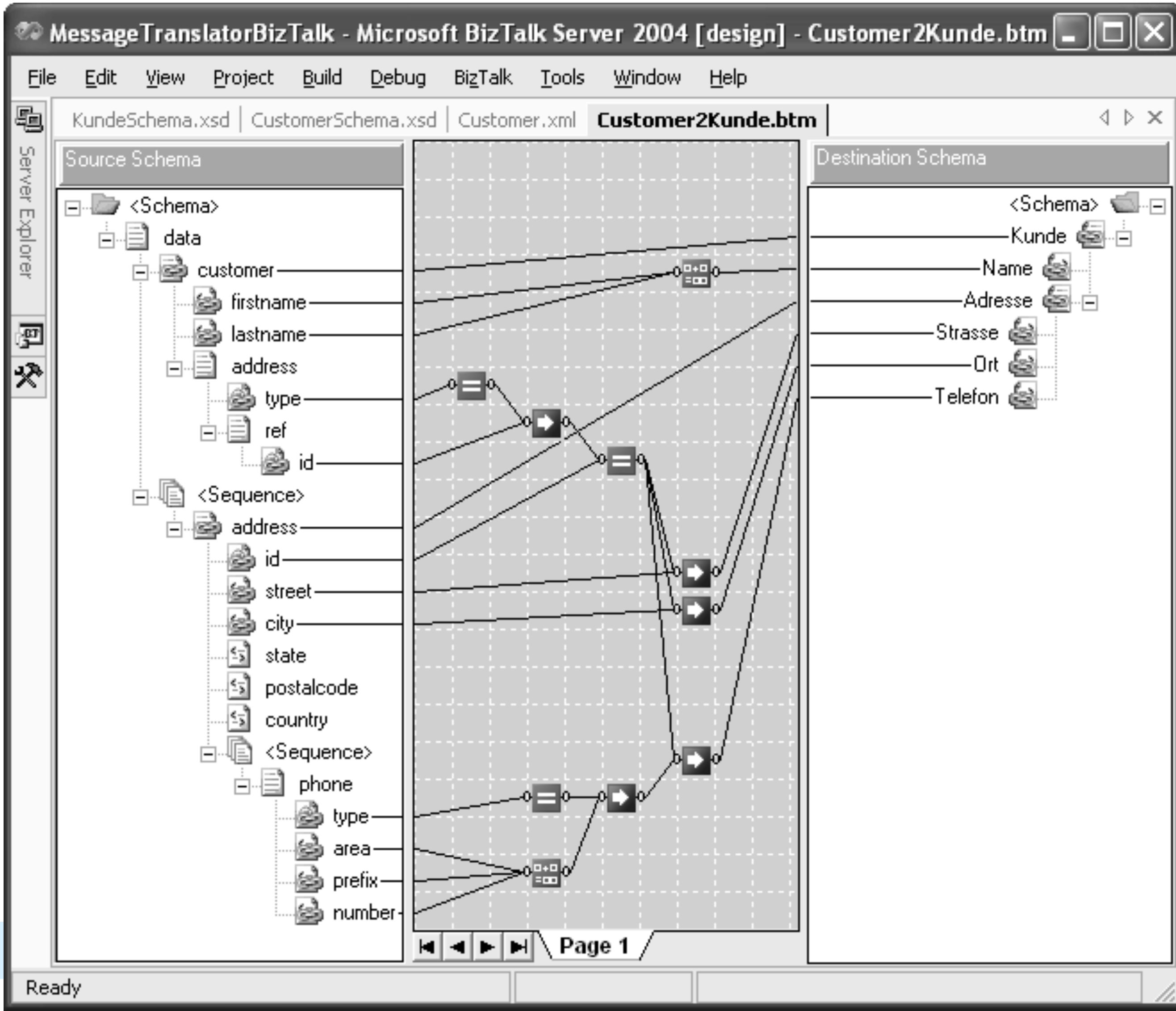
Levels of Transformation



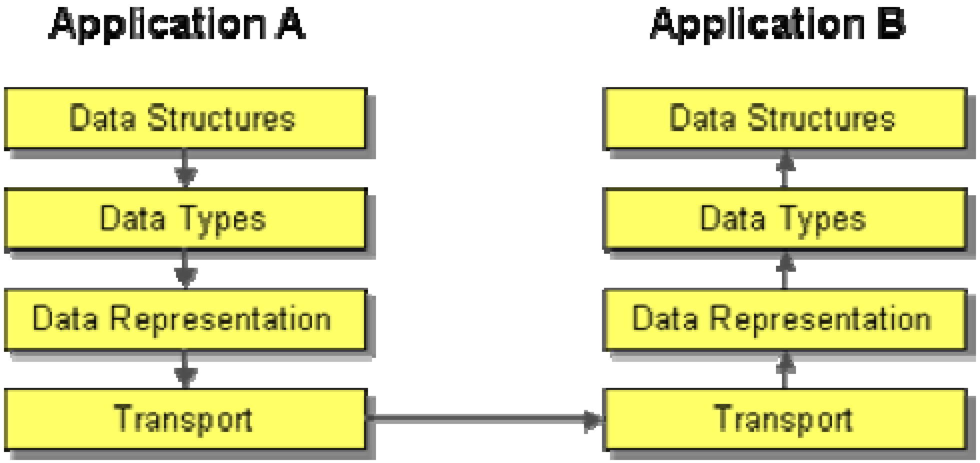
Layer	Deals With	Transformation Needs (Example)	Tools/ Techniques
Data Structures (Application Layer)	Entities, associations, cardinality	Condense many-to-many relationship into aggregation.	Structural mapping patterns, custom code
Data Types	Field names, data types, value domains, constraints, code 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

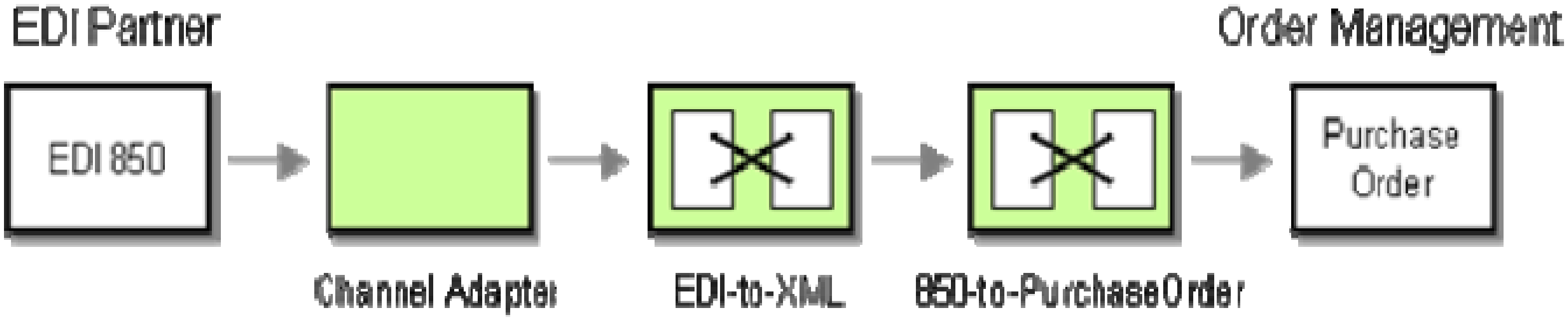
Visual transformation: drag-drop style

- Example:



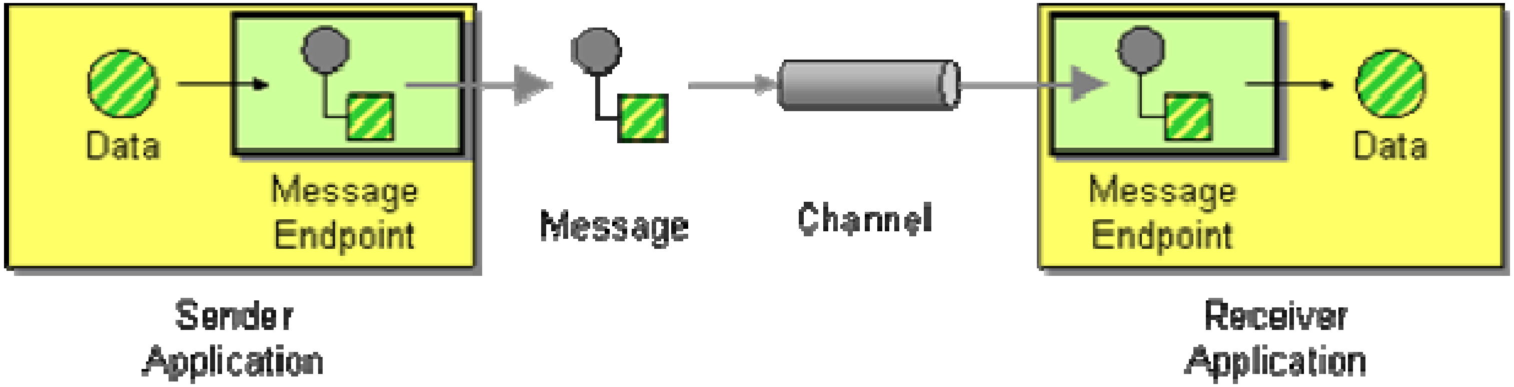
Chaining Transformations





Message Endpoint (95)

- ***How does an application connect to a messaging channel to send and receive messages?***
- Connect an application to a messaging channel using a *Message Endpoint,* a client of the messaging system that the application can then use to send or receive messages.



Message Endpoint …

- Is a specialized channel adapter - *custom developed* to interact with messaging system’s client API *)

- Encapsulates messaging system from the rest of app
  - Makes a message and sends it to messaging channel
  - Receives a message, extracts the contents, and gives it to the application

*) Should be designed as a Gateway (to hide the message system from the rest of the application)

Simple Messaging Coffee Shop Exercises

1. Download `MsgKit.zip` file from [github]

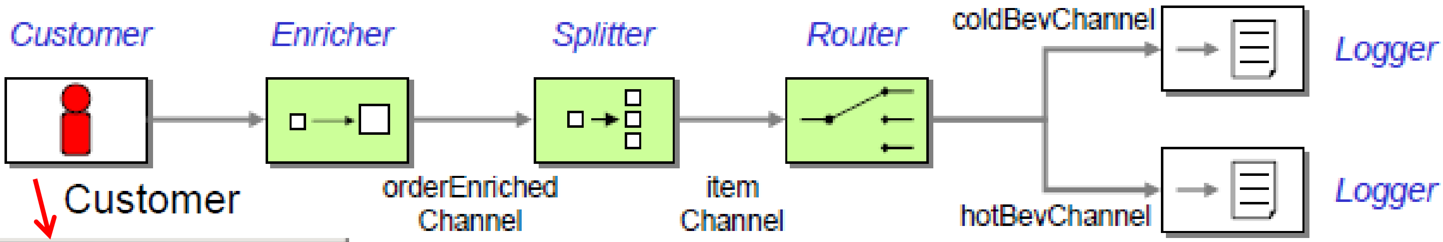
2. Follow Setup Instructions in `Simple Messaging Toolkit.pdf`

3. See `EIPTutorialReferenceChart.pdf` for overview of message components & tips about their use

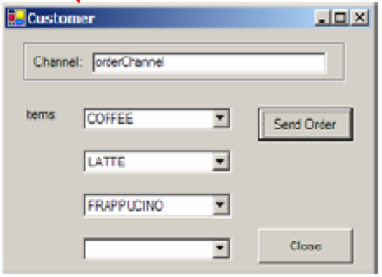
4. Make exercises 1-3 in `Simple Messaging Toolkit.pdf`
   - Use Messaging Domain Specific Language (see next slide)
   - Toolkit contains GUI components to visualize solution (see next slide)
   - Compose solutions in a batch file or the command line

Example of ToolKit Calls

```
call Customer orderChannel
call Enricher orderChannel orderEnrichedChannel
call Splitter orderEnrichedChannel itemChannel "/Order/Item"
call Router itemChannel coldBevChannel "Item = 'FRAPPUCINO'" hotBevChannel
call Logger coldBevChannel
call Logger hotBevChannel
```



Customer

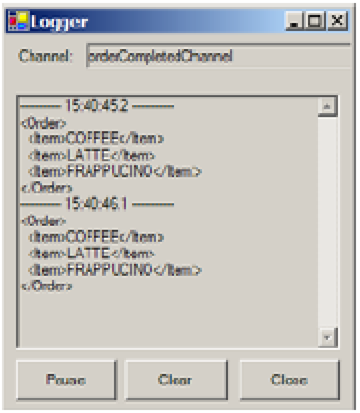


Channel: orderChannel

Items: COFFEE
LATTE
FRAPPUCINO

Sends order messages to specified channel

Logger

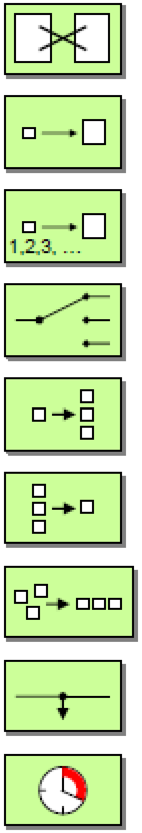


Channel: orderCompletedChannel

```
------- 15:40:45.2 -------
<Order>
  <Item>COFFEE</Item>
  <Item>LATTE</Item>
  <Item>FRAPPUCINO</Item>
</Order>
------- 15:40:46.1 -------
<Order>
  <Item>COFFEE</Item>
  <Item>LATTE</Item>
  <Item>FRAPPUCINO</Item>
</Order>
```

Display messages and time stamps

Available Pattern Components in ToolKit



- Message Translator
- Content Enricher (special case of Translator)
- Sequence Tagger
- Router
- Splitter
- Aggregator
- Resequencer
- Wire Tap (Tee)
- Delay

For Next Time (6/9)

- Finish Coffee Shop exercises
- Prepare for multiple choice test on Message Channels
- Read
  - EIP chap. 4: Message Channels
  - EIP chap. 5: Message Construction