Implementing Messaging Patterns in JavaScript using the OpenAjax Hub

Kevin Hakanson
Twin Cities Code Camp 11

Are You In The Right Place?

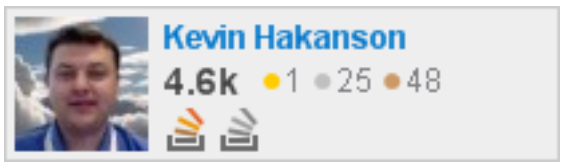
- This talk:
  - Is your web application a tightly coupled, DOM event handler mess? Use techniques from the Enterprise Integration Patterns book to build better components. Concepts including message, publish-subscribe channel, request-reply and message filter will be demonstrated in JavaScript (along with corresponding tests) using the OpenAjax Hub.
- This speaker:
  - Kevin Hakanson is an application architect for Thomson Reuters where he is focused on highly, scalable web applications. His background includes both .NET and Java, but he is most nostalgic about Lotus Notes. He has been developing professionally since 1994 and holds a Master's degree in Software Engineering. When not staring at a computer screen, he is probably staring at another screen, either watching TV or playing video games with his family.

Kevin Hakanson

@hakanson
#tccc11

kevin.hakanson@gmail.com

github.com/hakanson/tccc11



stackoverflow.com/users/22514/kevin-hakanson

What to Expect

- "Based on a True Story"
  - derived from production code from large scale, web app
- Enterprise Integration Patterns
  - selected patterns from the book
- JavaScript Libraries
  - OpenAjax, jQuery, QUnit
- Problems + Patterns + Tests
- Living on the edge
  - Google docs presentation, Cloud9 IDE, github
- Questions (OK during presentation)

Let's Get Started!

Problem

- large scale web application
- multiple teams with separate code bases
- iterative development, but long duration project
- continuous integration, build/deploy on check in
- primary touch point is in browser
- automated regression tests

- make all this work without app constantly breaking

Architect Talk

- Treat UI components (widgets) like services in the browser which operate independent of each other
  - however, some have no UI and are invisible services
- Think in terms of application level messages between components instead of direct wiring/capturing another components raw DOM events
  - internal component interaction can be DOM or custom event based
- Hierarchical channels with naming standard and wildcard subscriptions. Long lived and should be namespaced.
- Forwards and backwards version compatibility

insert "Ivory Tower Architect" joke here :)

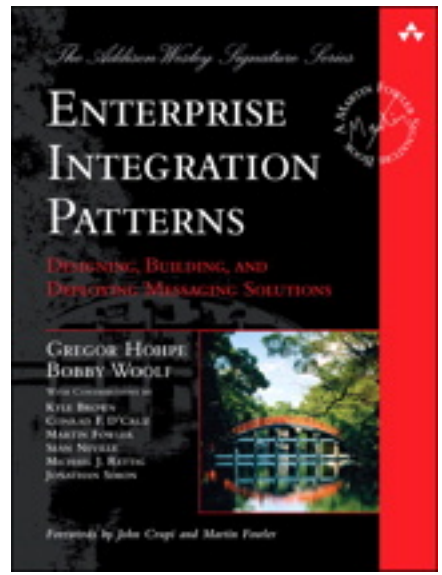
Demo

```
$( ".slide[title='Demo']" ).focus( function() {
    window.location.href = "demo.html";
});
```

Enterprise Integration Patterns

Enterprise Integration Patterns:
Designing, Building, and Deploying Messaging Solutions

By Gregor Hohpe, Bobby Woolf
Published Oct 10, 2003 by Addison-Wesley Professional. Part of the Addison-Wesley Signature Series (Fowler) series.
ISBN-10: 0-321-20068-3
ISBN-13: 978-0-321-20068-6



http://www.enterpriseintegrationpatterns.com/

List of Patterns

Messaging Systems:

- Message Channel, Message, Pipes and Filters, Message Router, Message Translator, Message Endpoint

Messaging Channels:

- Point-to-Point Channel, Publish-Subscribe Channel, Datatype Channel, Invalid Message Channel, Dead Letter Channel, Guaranteed Delivery, Channel Adapter, Messaging Bridge, Message Bus

Message Construction:

- Command Message, Document Message, Event Message, Request-Reply, Return Address, Correlation Identifier, Message Sequence, Message Expiration, Format Indicator

Message Routing:

- Content-Based Router, Message Filter, Dynamic Router, Recipient List, Splitter, Aggregator, Resequencer, Composed Msg. Processor, Scatter-Gather, Routing Slip, Process Manager, Message Broker

Message Transformation:

- Envelope Wrapper, Content Enricher, Content Filter, Claim Check, Normalizer, Canonical Data Model

Messaging Endpoints:

- Messaging Gateway, Messaging Mapper, Transactional Client, Polling Consumer, Event-Driven Consumer, Competing Consumers, Message Dispatcher, Selective Consumer, Durable Subscriber, Idempotent Receiver, Service Activator

System Management:

- Control Bus, Detour, Wire Tap, Message History, Message Store, Smart Proxy, Test Message, Channel Purger

JavaScript Publish-Subscribe Options

- jQuery custom events (api.jquery.com/category/events/)
  - flat name with optional, restricting namespace
  - reflects DOM structure, including event bubbling
  - API: bind, unbind, trigger
- AmplifyJS Pub/Sub (amplifyjs.com/api/pubsub/)
  - cleaner interface that jQuery custom events
  - subscriptions can request priority and/or prevent other callbacks from being invoked
  - API: subscribe, unsubscribe, publish
- OpenAjax Hub 1.0 (www.openajax.org)
  - simple and lightweight publish/subscribe engine
  - hierarchical event names with wildcard subscriptions
  - API: subscribe, unsubscribe, publish

QUnit

"QUnit is a powerful, easy-to-use, JavaScript test suite. It's used by the jQuery project to test its code and plugins but is capable of testing any generic JavaScript code."
http://docs.jquery.com/Qunit

QUnit Example

```
test("a basic test example", function() {
  ok( true, "this test is fine" );
  var value = "hello";
  equal( value, "hello", "We expect value to be hello" );
});
```

Tests

- QUnit tests of jQuery custom events
  - https://github.com/hakanson/tccc11/blob/master/tests/jquery.tests.js
  - http://jsfiddle.net/hakanson/5bgjd/
- QUnit tests of AmplifyJS Pub/Sub
  - https://github.com/hakanson/tccc11/blob/master/tests/amplify.tests.js
  - http://jsfiddle.net/hakanson/zy6Dy/

$.hub

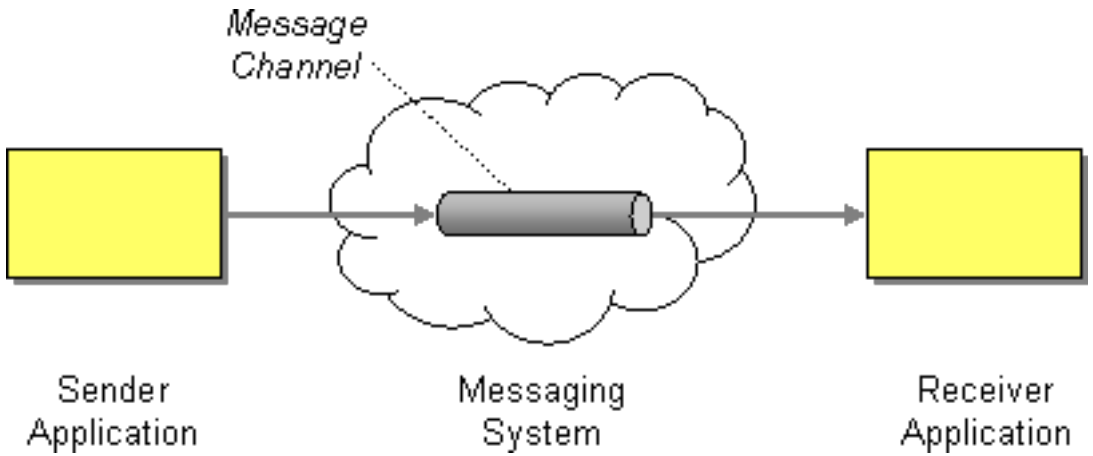
- jQuery plug-in created for this talk
  - derived from production code
- Builds on top of OpenAjax Hub functionality
- API:
  - subscribe, unsubscribe, publish
  - guid, message
  - createVersionFilter, createIdempotentFilter
  - reply, requestReply

Problem (again)

- make all this work without app constantly breaking

Message Channel (60)

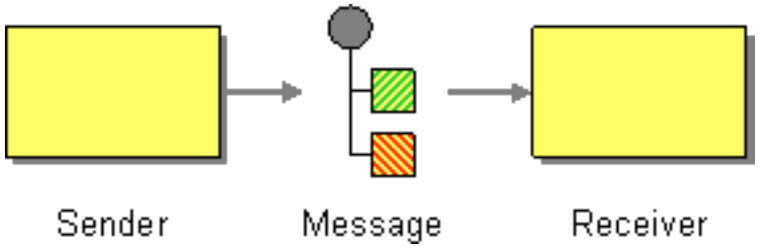
How does one application communicate with another using messaging?



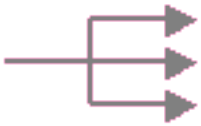
Connect the applications using a *Message Channel*, where one application writes information to the channel and the other one reads that information from the channel.

Message (66)

How can two applications connected by a message channel exchange a piece of information?

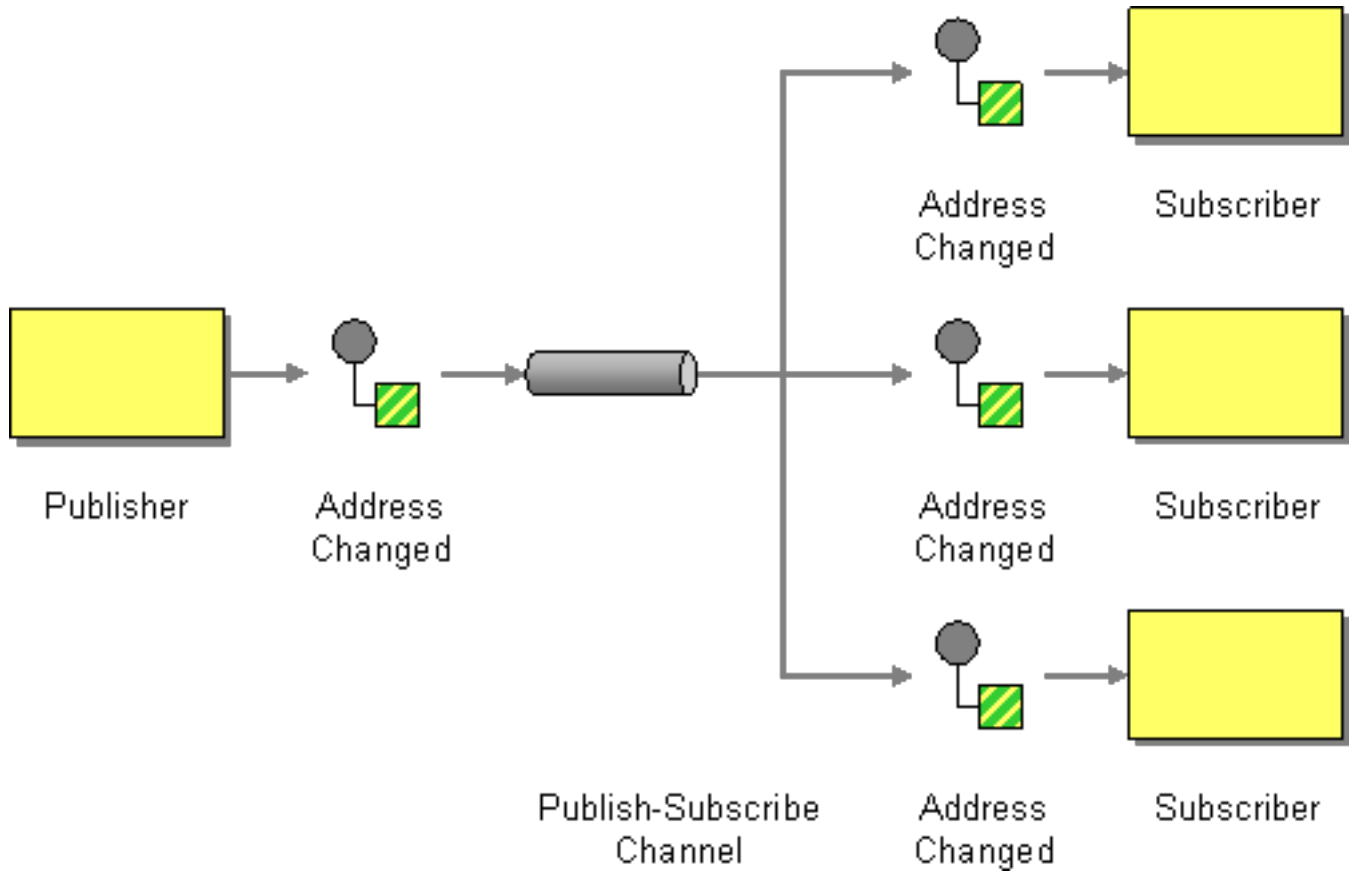


Package the information into a *Message*, a data record that the messaging system can transmit through a message channel.



Publish-Subscribe Channel (106)

How can the sender broadcast an event to all interested receivers?



Send the event on a *Publish-Subscribe Channel*, which delivers a copy of a particular event to each receiver.

channel parameter

- the name of the event to listen for
  - OpenAjax - name
  - AmplifyJS - topic
  - jQuery custom events - eventType
  - elsewhere - subject

message parameter

- An arbitrary Object holding extra information that will be passed as an argument to the handler function
  - OpenAjax - publisherData
  - jQuery custom events - extraParameters
  - AmplifyJS - additional parameters

callback parameter

- a function object reference to be called when a subscription receives data
- should play nice and handle:
  - own errors and not throw exceptions
  - being called synchronous and not perform long running operations
  - being called asynchronous (depending on implementation)

- scope parameter
  - what this is when callback is invoked (default window)
  - AmplifyJS - context
  - or use callback, scope with $.proxy(function, context)

Test Excerpt

```
$.hub.subscribe( "A", function( channel, message ) {
    countA++;
});
$.hub.subscribe( "A.*", function( channel, message ) {
    countAstar++;
});
$.hub.subscribe( "A.**", function( channel, message ) {
    countAstarstar++;
});
$.hub.subscribe( "A.1", function( channel, message ) {
    countA1++;
});
$.hub.subscribe( "A.1.a", function( channel, message ) {
    countA1a++;
});
$.hub.publish( "A", message );
$.hub.publish( "A.1", message );
$.hub.publish( "A.1.a", message );
$.hub.publish( "A.1.b", message );
```

Demo (Updated)

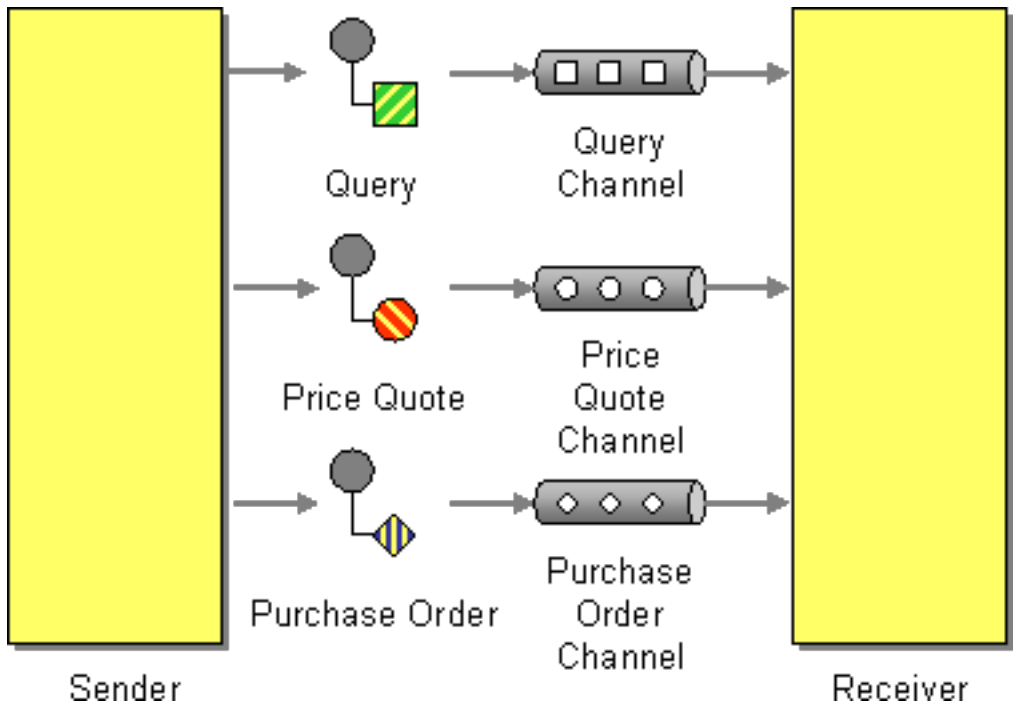
```
$.get("demo.html", function(data) {
  data.replace( "demo.js", "demo-updated.js" );
});
```

Architect Talk

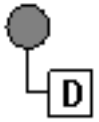
- If messages published on a channel carry same type of body, then channel names identify types of messages
  - Message is channel + headers + body
- Message body carries data payload; defined by specification
  - in future could use XML or JSON schema
- Message should be serializable (JSON) and not contain functions or logic
  - better for immutability/tampering and cross domain usages (iframes, client to server)

Datatype Channel (111)

How can the application send a data item such that the receiver will know how to process it?

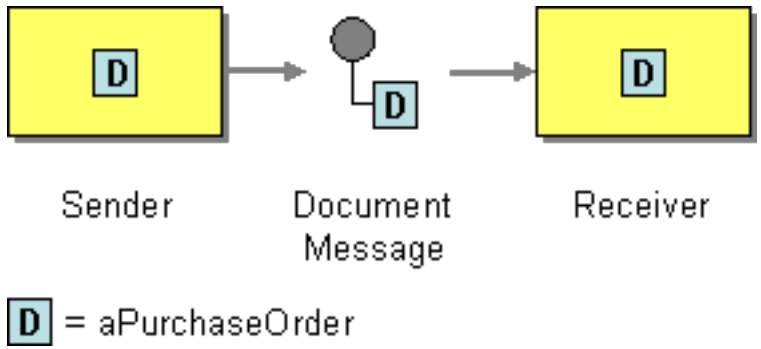


Use a separate *Datatype Channel* for each data type, so that all data on a particular channel is of the same type.



Document Message (147)

How can messaging be used to transfer data between applications?



Use a *Document Message* to reliably transfer a data structure between applications.

$.hub.message


```
{
   "timestamp": 1317961214398,
   "messageId": "31525258-8ca4-464f-96da-668e52a6cad9",
   "formatVersion": "0",
   "body": {
      "key1": "value1",
      "key2": "value2"
   }
}
```

Test Excerpt

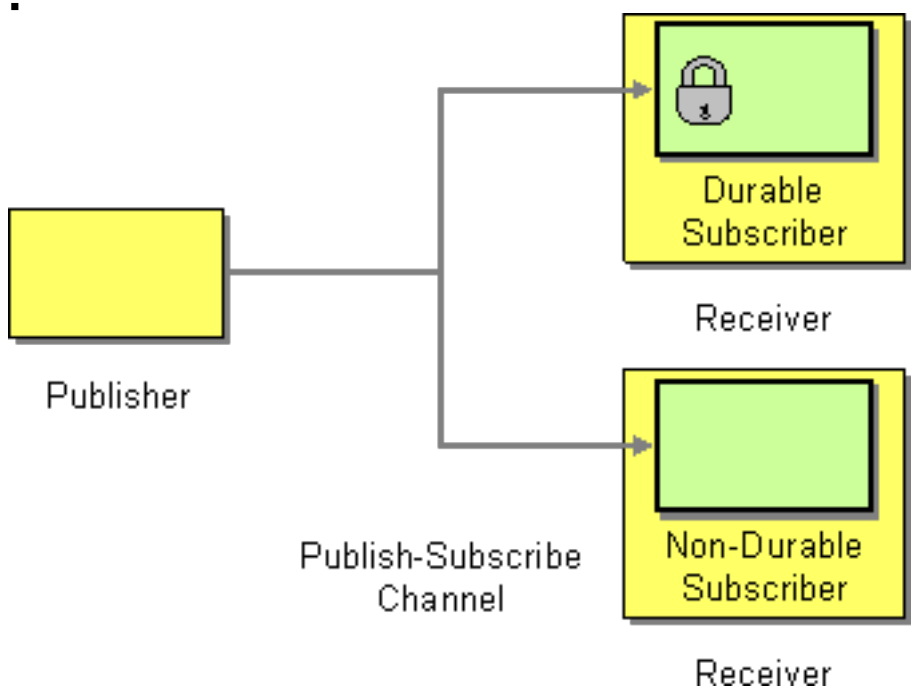
```
var channel = "name",
    messageBody = { "key1": "value1", "key2": "value2"},
    actualMessage,
    actualChannel;

var subscription = $.hub.subscribe( channel,
    function( channel, message ) {
        actualChannel = channel;
        actualMessage = message;
    });

var message = $.hub.message( messageBody );

$.hub.publish( channel, message );

$.hub.unsubscribe( subscription );
```

Problem

- Two components exist on a web page
  - first component subscribes to messages from the second component
  - second component publishes message during initialization
- When <script> tags are reordered, the functionality breaks because second component publishes its message before first component has a chance to subscribe
  - components on page have temporal coupling
- Need the equivalent of jQuery $.ready() for hub

Durable Subscriber (522)

How can a subscriber avoid missing messages while it's not listening for them?



Use a *Durable Subscriber* to make the messaging system save messages published while the subscriber is disconnected.

Test Excerpt

```
// tell hub to save messages
$.hub({ ready: false });

// publish before any active subscribers
$.hub.publish( channelA, messageA );

var subscriptionA = $.hub.subscribe(channelA,
    function( channel, message ) {
        actualChannel = channel;
        actualMessage = message;
    });

// tell hub to delivery saved messages
$.hub({ ready: true });
```

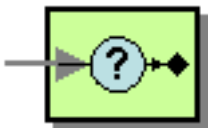
Architect Talk

- Need message body to be backwards compatible in both structure and meaning
  - otherwise need versioning scheme and strategy for both old and new versions of a message to be published
- A filter on version is important for backwards and forwards compatibility, so your code doesn't break when a new version of the message is published
  - using the filter parameter allows you to put the guard outside of the callback instead of inside it (think of the filter like an aspect)

Format Indicator (180)

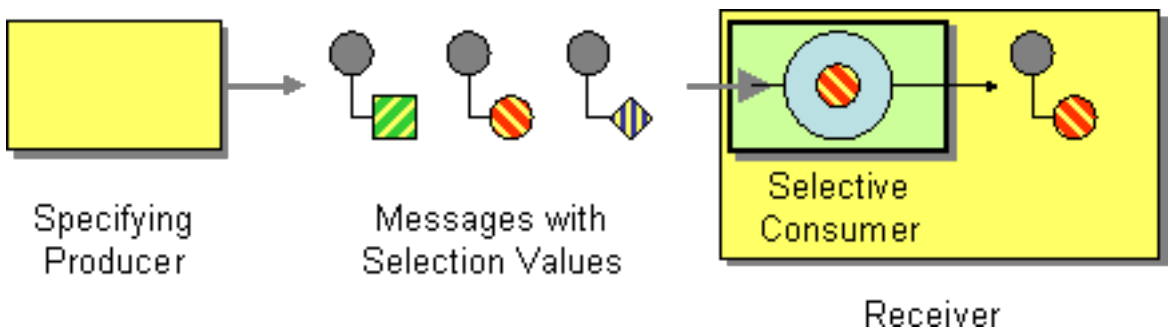
How can a message's data format be designed to allow for possible future changes?

Design a data format that includes a *Format Indicator*, so that the message specifies what format it is using.



Selective Consumer (515)

How can a message consumer select which messages it wishes to receive?



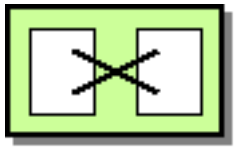
Make the consumer a *Selective Consumer*, one that filteres the messages delivered by its channel so that it only receives the ones that match its criteria.

filter parameter

- A function that returns true or false to either match or deny a match of the published event

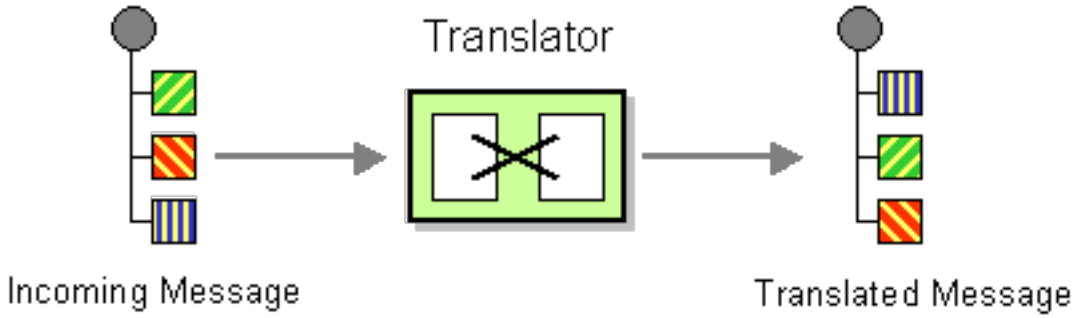
Test Excerpt

```
var m1 = $.hub.message({ name: "Kevin Hakanson" },
    { formatVersion: "1" });
var m2 = $.hub.message({ firstname: "Kevin", lastname: "Hakanson" },
    { formatVersion: "2" });

function callbackV1( channel, message ) {
    countV1++;
}
function callbackV2( channel, message ) {
    countV2++;
}

$.hub.subscribe( "Person.*", callbackV1, null, null,
    $.hub.createVersionFilter( "1" ) );
$.hub.subscribe( "Person.*", callbackV2, null, null,
    $.hub.createVersionFilter( "2" ) );

$.hub.publish( "Person.Cool", m1 );
$.hub.publish( "Person.Cool", m2 );
```



Message Translator (85)

How can systems using different data formats communicate with each other using messaging?



Use a special filter, a *Message Translator*, between other filters or applications to translate one data format into another.

Test Excerpt

```
var m2 = $.hub.message({ firstname: "Kevin", lastname: "Hakanson" },
    { formatVersion: "2" });

function callbackV1(channel, message ) {
    countV1++;
}
function callbackV2(channel, message ) {
    countV2++;
    var name = message.body.firstname + " " + message.body.lastname;
    var translatedMessage = $.hub.message( { name: name },
        { formatVersion: "1" } );
    $.hub.publish( channel, translatedMessage );
}

subscriber1 = $.hub.subscribe( "Person.*", callbackV1, null, null,
    $.hub.createVersionFilter( "1" ) );
subscriber2 = $.hub.subscribe( "Person.*", callbackV2, null, null,
    $.hub.createVersionFilter( "2" ) );

$.hub.publish( "Person.Cool", m2 );
```

Idempotent Receiver

How can a message receiver deal with duplicate messages?

Design a receiver to be an *Idempotent Receiver*--one that can safely receive the same message multiple times.

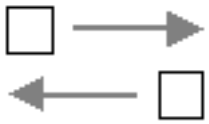
Test Excerpt

```
var m1 = $.hub.message({}),
    m2 = $.hub.message({});

function callback1( channel, message ) {
    count1++;
}
function callback2( channel, message ) {
    count2++;
}

$.hub.subscribe( channel, callback1, null, null, createIdempotentFilter() );
$.hub.subscribe( channel, callback2, null, null, createIdempotentFilter() );

$.hub.publish( channel, m1 );
$.hub.publish( channel, m1 );
$.hub.publish( channel, m2 );
$.hub.publish( channel, m2 );
$.hub.publish( channel, m2 );
```

Problem

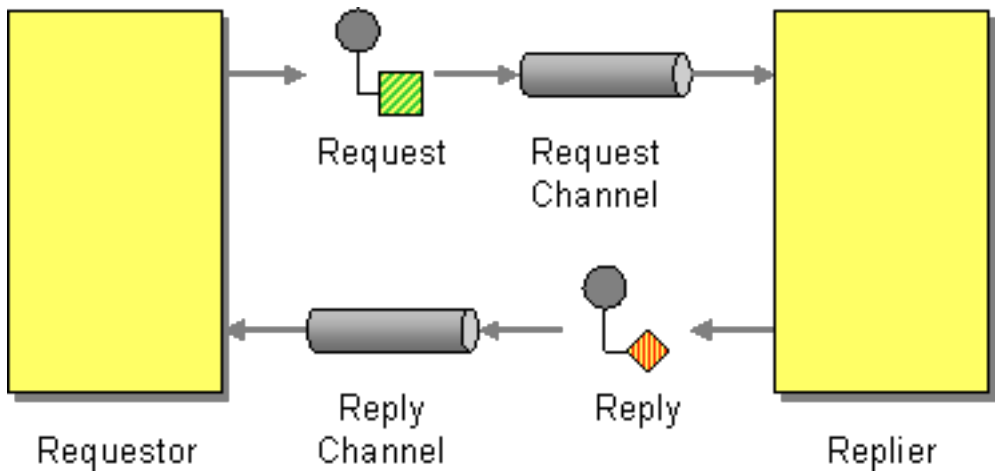
- How to replace directly calling functions with messaging?
- Want to do something like below, without common coupling (sharing a global variable).
  - var component1 = new Component1();

    component1.command();
    var reply = component1.request({});



Request-Reply (154)

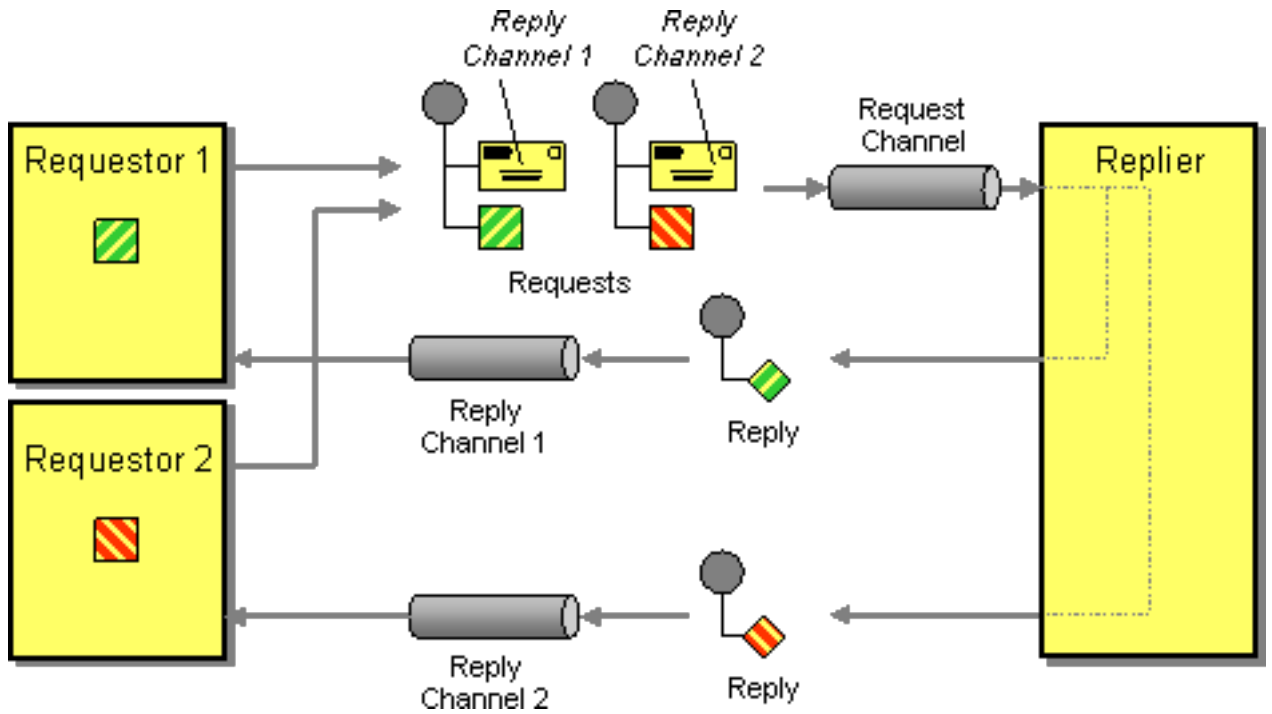
When an application sends a message, how can it get a response from the receiver?



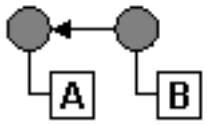
Send a pair of *Request-Reply* messages, each on its own channel.

Return Address (159)

How does a replier know where to send the reply?

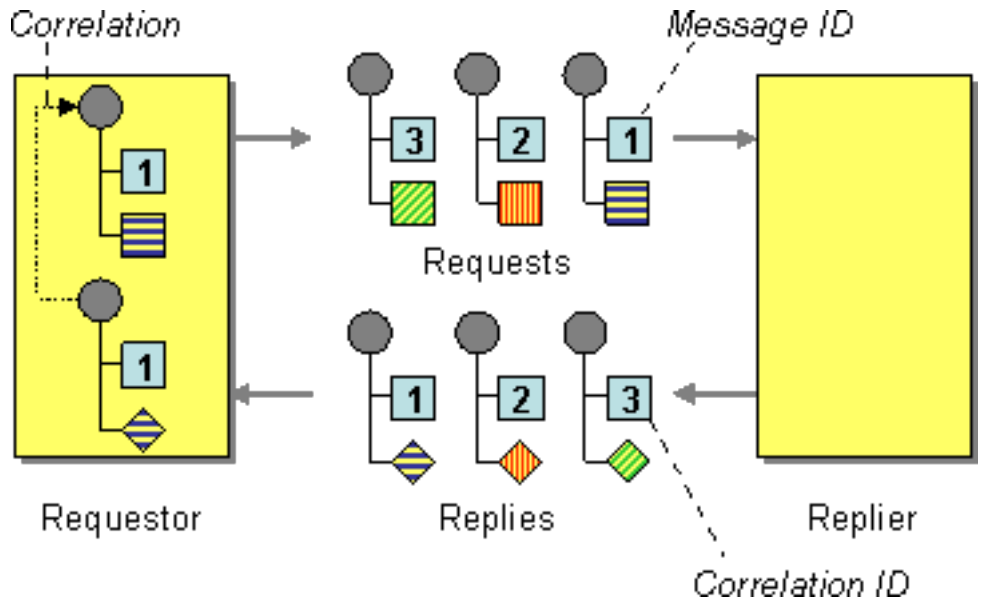


The request message should contain a *Return Address* that indicates where to send the reply message.



Correlation Identifier (163)

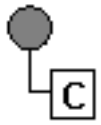
How does a requestor that has received a reply know which request this is the reply for?



Each reply message should contain a *Correlation Identifier*, a unique identifier that indicates which request message this reply is for.

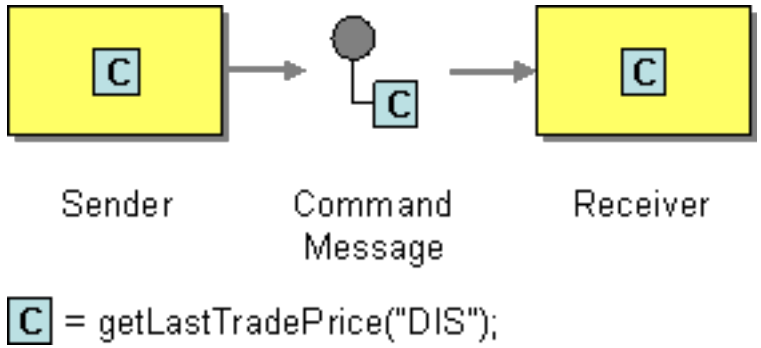
Test Excerpt

```
$.hub.subscribe(channel, function( channel, message ) {
    actualChannel = channel;
    actualMessage = message;
    $.hub.reply( message, replyMessage );
});

$.hub.requestReply(channel, message,
    function( channel, message ) {
        actualReplyMessage = message;
        correlationId = message.correlationId;
    });
```

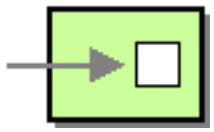


Command Message (145)

How can messaging be used to invoke a procedure in another application?

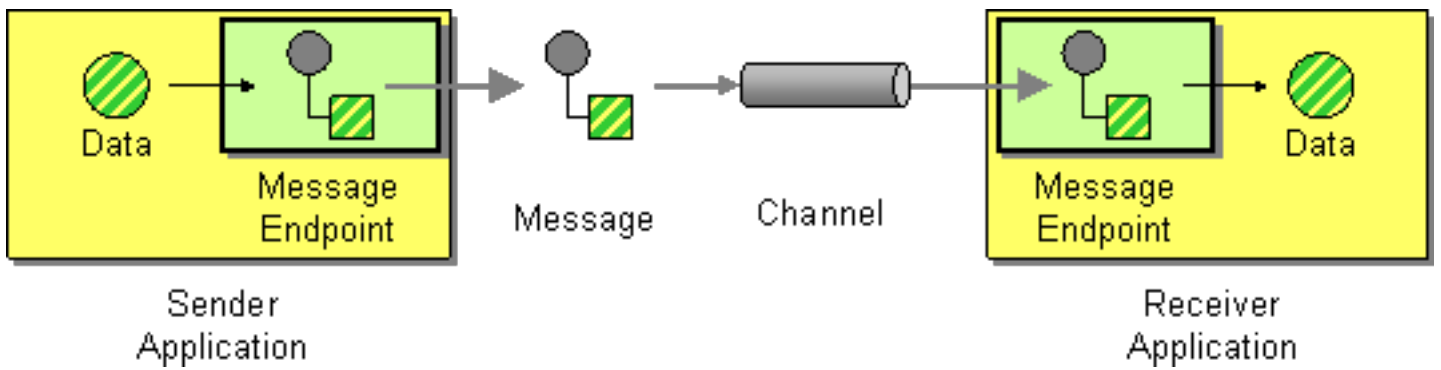


Use a *Command Message* to reliably invoke a procedure in another application.



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.

Test Excerpt

```
// create a message endpoint to call counter.increment()
var subscription = $.hub.subscribe( channel,
    function( channel, message ) {
        counter.increment( message );
    });

counter.increment( 2 );

// send command message to increment by 2
$.hub.publish( channel, 2 );
```

Summary

- Patterns help us find appropriate solutions in the software architecture and design process.
- Using Publish/Subscribe events for browser communication adds a level of indirection between the JavaScript components.
- Using a common messaging structure for the events (including naming and versioning) make things more predictable.
- Libraries are not enough; knowledge of JavaScript and the DOM is essential for good widget design and interaction.

Q & A