

Fearless International (FRLE) \$0.19

Fearless International Inc., a luxury performance boat manufacturer, has been the focus of the media for the last several months in magazine such as GQ, Time, Bloomberg Markets, Maxim, and over 20 others.

According to TIME, "When a company bold enough to call itself Fearless Yachts splashed onto the luxury-boat market, it drew considerable attention. Collaborating with Porsche Design Studio/Austria on a series of high-style, high-performance yachts, the brand unveiled its first model, the Fearless 28, in February." Since its release, Fearless Yachts has taken orders for more than 33 Yachts bringing more than \$10 Million in sales and put the production facility at 75% capacity.

Top 5 Reasons To Consider Fearless Yachts:

- 1: Already \$10,000,000 In Sales Since First 7 Months.
- 2: First of a 5 yacht series had huge response from the market.
- 3: Next yacht designs have been released and Debut is in Miami in February
- 4: Company set to begin international marketing.

More coupling gives us efficiency and simplicity but can also introduce rigidity and brittleness.

The remainder of the code is little more than a loop over an XML node set.

It also means that often we can look at daily life to help design successful messaging solutions.

It allows them to deploy multiple baristas in a Competing Consumer scenario if t

he store gets busy.

A pattern applies most of the time, but not always.

Many tools and libraries embody many successful patterns.

This does not mean that patterns cannot be incorporated into a tool.

This means that the behavior of the pattern can be fixed as long as the expressions are variable.

The diagram shows a simple example of a customer placing an order, which is split into individual order items by a Splitter.

It can be useful to distinguish levels of sophistication for subscribers to express their subscription preferences: Simple Topics.

Essentially, the messaging infrastructure implements an Aggregator.

This might in fact be useful in larger systems.

When a processing element receives an event message it perform its designated function and publishes one or more resulting events.

This implies that we need to think carefully about validating the composability as well as the actual composition of the individual processing elements.

However, we have to keep in mind that such architectures introduce an additional layer, the composition layer, into the system.

One definite contributor to the easy conversion of these patterns into components is the Pipes-and-Filters architectural style.

Using such a two-phase-commit approach would certainly kill Starbucks' business because the number of customers they can serve within a certain time interval would decrease dramatically.

Or, the type of message on a channel might not match what the element expects.

Let it suffice to say that in my opinion coupling is a measure of dependency between systems.

However, most implementations implement this functionality in the receiver using a reactive filtering approach as opposed to in the messaging infrastructure.

His mission is to make integration and distributed system development easier by harvesting common patterns and best practices from many different technologies.

Along the way orders can be enriched, split, aggregated, resequenced etc.

Or, the type of message on a channel might not match what the element expects.

Cynics might say that patterns book only contain stuff you already knew while research papers are never interested in something that is actually useful in practice.

They do so by placing entries in form of tuples into the blackboard.

If a message is identified by a number of independent fields and values, this mapping can be unnatural.

Almost every messaging library supports Competing Consumers, Event-driven Consumers and many more patterns.

mapped into a specific programming environment to be used.

Is it valid to publish an event that no one subscribes to?

Because the state of the conversation is represented by a message sitting in a specific message queue, each message queue might have to be inspected.

Every modern computer has an instruction counter that points to the next instruction to be executed.

We call these elements variability points.

This might in fact be useful in larger systems.

So, chances you are whatever you build outside of an op-amp, it is stateful.

In an asynchronous conversation, each conversation partner has to "remember" what to do next once a message comes in.

What if I need a reply back?

Content-based Subscription.

First, JavaOne et al often degrade into a series of advertorials so it is refreshing to see less snazzy but more information rich presentations.

This can quickly lead to a huge number of open connections, something a TCP stack is usually not designed to handle.

This might seem like a bad plan but in the reality of business this option might be acceptable.

So what is the secret sauce here?

A pattern applies most of the time, but not always.
These events can then be picked up by other processing nodes.
SeeBeyond got so much crap from the analysts for its Monk language.
Voila, we have converted a pattern into a generic enough component.
Because the state of the conversation is represented by a message sitting in a specific message queue, each message queue might have to be inspected.
For example, a message might be sent to the topic "prod."
This implies that we need to think carefully about validating the composability as well as the actual composition of the individual processing elements.
The downside is that a shared blackboard can become a bottleneck in a large distributed system.
The downside is that we use the TCP stack as a surrogate call stack for a distributed system with long-running interactions.
customer", such as "prod."
All of these strategies are different than a two-phase commit that relies on separate prepare and execute steps.
The revenue loss was small enough to allow the business to operate in this way.
The sole role of this coordinator is to track the conversation across the multiple parties.
Essentially, the messaging infrastructure implements an Aggregator.
Once we get this part out of our system, the debate of statelessness becomes a lot more productive.
Codifying Patterns Typically, when people ask me about "codifying" or "toolifying" patterns my first reaction is one of skepticism.
For example, a conversational state could be that component A has made a request and is now waiting for the reply.
A special case is a retry with Idempotent Receiver.
New Programming Model Testing the actual composition of the individual elements is another important part of validating the system.
If you disagree you are welcome to argue with me, but you are buying!
However, the system automatically preprocesses the information such that it is easily understood by human beings.
More coupling gives us efficiency and simplicity but can also introduce rigidity and brittleness.
For example, if a business rule is violated it is unlikely a retry will succeed.

While waiting for my "Hotto Cocoa" I started to think about how Starbucks processes drink orders.
This requires us to rethink our approach to testing slightly.
Every modern computer is stateful.
Composability One of the advantages of composability is that the processing elements can be reused in various constellations because the elements do not make any direct assumptions about each other.
But we can be assured that once SOAs reach wide adoption all marketing departments are itching to define the next major trend.
Are "Pattern" and "Component" Antonyms?
Also, this approach often makes it inefficient to monitor the state of a conversation.
They do so by placing entries in form of tuples into the blackboard.
Compensating Action - The last option is to undo operations that were already completed to put the system back into a consistent state.
This approach makes for a highly scalable message flow architecture, however, real life constraints often force a compromise on this pure architectural style.
It's the context-problem-forces-solution combination that makes patterns so useful.
As a result I think that the bar to adoption of new languages in commercial environments is quite high.
We call these elements variability points.
The blackboard will return a reference to any entry that matches the request.

What if I don't have all required information available to assemble a single document?

So where could part of this conversational state be kept?

It is not relevant to the discussion at the chosen level of abstraction.

Compensating Action - The last option is to undo operations that were already completed to put the system back into a consistent state.

The advantage is that the application process does not have to worry about correlation.

First, I fixed a lot of the technical environment parameters: the components only run on a .

We call these elements variability points.

Using such a two-phase-commit approach would certainly kill Starbucks' business because the number of customers they can serve within a certain time interval would decrease dramatically.

They will toss the drink if it has already been made or otherwise pull your cup from the "queue".

As a result, Starbucks has a correlation problem.

Correlation By taking advantage of an asynchronous approach Starbucks also has to deal with the same challenges that asynchrony inherently brings.

Using such a two-phase-commit approach would certainly kill Starbucks' business because the number of customers they can serve within a certain time interval would decrease dramatically.

So, chances you are whatever you build outside of an op-amp, it is stateful.

Naturally, it is difficult to test explicitly for unforeseen scenarios.

For example, complex conversations almost always require some state to be kept in the endpoints or an explicit conversation coordinator.

Correlation By taking advantage of an asynchronous approach Starbucks also has to deal with the same challenges that asynchrony inherently brings.

The variability point is how the router makes the decision, for example which field of the incoming message it looks at.

For example, a conversational state could be that component A has made a request and is now waiting for the reply.

As a result, a customer might end up with active service but would not get billed.

A Logger displays messages on the orderCompletedChannel.

Integration Testing As a result, integration testing is more critical than ever with loosely coupled, composable systems.

Every modern computer has an instruction counter that points to the next instruction to be executed.

Because the domain-specific language is interpreted it is easy to allow it to be specified on the command line.

It is also equally hotly debated.

A Logger displays messages on the orderCompletedChannel.

For example, if a business rule is violated it is unlikely a retry will succeed.

For example, we might have composed something that is perfectly valid according to the loosely coupled architecture but that does not actually do anything useful.

Many tools and libraries embody many successful patterns.

If the loss is small it might be more expensive to build an error correction solution than to just let things be.

Finally the money, receipt and drink would change hands in one swoop.

Every modern computer has an instruction counter that points to the next instruction to be executed.

However, most implementations implement this functionality in the receiver using a reactive filtering approach as opposed to in the messaging infrastructure.

This type of publish-subscribe messaging is quite common, for example in TIBCO Rendezvous.

So what is the secret sauce here?

It is not relevant to the discussion at the chosen level of abstraction.

Periodically, they would run reconciliation reports to detect the "free" accounts and close them.

This type of publish-subscribe messaging is quite common, for example in TIBCO Rendezvous.

In the Endpoints In order to track more complex conversations a portion of the conversation state is typically kept in the communication endpoints.

Neither one is a trivial task.

The goal of pattern authors is to find common themes in existing usage and to make them easily understood.

Many of the routing patterns are very close descendants of the basic router.

The free composability makes it interesting in fun to build larger messaging solutions from these relatively simple building blocks.

So what were some of the current research topics that were discussed?

Conclusion it was fun to attend a more academic conference for a change.

Hidden assumptions are often unavoidable but can also introduce a glut of problems.

Voila, we have converted a pattern into a generic enough component.

If a message is identified by a number of independent fields and values, this mapping can be unnatural.

A topic tree forces all messages to be attached to a hierarchy.

Cynics might say that patterns book only contain stuff you already knew while research papers are never interested in something that is actually useful in practice.

Is it valid to have multiple subsystems that communicate amongst themselves but not between each other?

But we can be assured that once SOAs reach wide adoption all marketing departments are itching to define the next major trend.

Once the architectural style is expressed more formally, one can envision actual languages that can help express the desired constraints.

SeeBeyond got so much crap from the analysts for its Monk language.

For example, a processing element may listen for events that are never published.

A Logger displays messages on the orderCompletedChannel.

So where could part of this conversational state be kept?

Message Selectors allow subscribers to specify an expression to apply against incoming messages.

However, the actual system exhibits fatal flaws, a popular one being a system that does plain nothing.

Cynics might say that patterns book only contain stuff you already knew while research papers are never interested in something that is actually useful in practice.

This implementation is naturally inefficient because a message has to be delivered to the endpoint just to find out that the message does not match the expression.

Naturally, there are a number of issues to be resolved.

In the US, most Starbucks use an explicit correlation identifier by writing your name on the cup and calling it out when the drink is complete.

Conversations The coffee shop interaction is also a good example of a simple, but common Conversation pattern.

It turns out that the variability points for most routing patterns are relatively simple expressions.

As a result they use asynchronous processing.

Research Papers After I finished teaching my tutorial someone came up and essentially said that the patterns are not that useful for their work because they are mostly old stuff.

As a result, yet another TCP connection is established and held for the duration of this request-reply interaction.

A Conversation is a sequence of coordinated message exchanges.

Also, it might keep internal state data to be carried over to the processing of the reply message.

So any debate of statelessness is futile unless a clear level of abstraction is set and agreed upon first.

The key is whether the statefulness matters at our level of abstraction.

Most of the time this happens because the individual components were improperly composed.

Integration Testing As a result, integration testing is more critical than ever with loosely coupled, composable systems.

It also means that often we can look at daily life to help design successful messaging solutions.

TCP Stack One equally popular and inefficient way to represent the state of the conversation is the TCP stack.

Still, I like to attend more academic conferences from time to time.

In a multi-party conversation each individual participant may only have a partial view of the conversation.

Also, it might keep internal state data to be carried over to the processing of the reply message.

The simplest way to subscribe to messages is to specify a topic.

New Programming Model Testing the actual composition of the individual elements is another important part of validating the system.

It turns out that the variability points for most routing patterns are relatively simple expressions.

Validation Rules Testing in the traditional sense will only perform part of the necessary validation.

Conclusion it was fun to attend a more academic conference for a change.

Because the state of the conversation is represented by a message sitting in a specific message queue, each message queue might have to be inspected.

David Orchard has given us a nice definition of the ultimate loosely coupled solution by stating: "How do you make two systems loosely coupled?"

Distinguishing what kind of state is kept where and whether this state is relevant is an important first step to having a debate related to this topic.

The TCP stack is chartered with correlating incoming response data packets to the right process waiting for a response.

The advantage is that the application process does not have to worry about correlation.

More coupling gives us efficiency and simplicity but can also introduce rigidity and brittleness.

Wildcards are a simple enhancement to simple topics.

Not because Monk is bad but because of the cost and risk of training developers in a new language.

A Publish-Subscribe Channel sends a copy of a message to multiple recipients, based on the subscription preferences of the subscribers.

In the Endpoints In order to track more complex conversations a portion of the conversation state is typically kept in the communication end points.

The key is whether the statefulness matters at our level of abstraction.

Most of the time this happens because the individual components were improperly composed.

Many of the routing patterns are very close descendants of the basic router.

Instead, we often have to use code analysis or inspection to make any statements about composability.

Research Papers After I finished teaching my tutorial someone came up and essentially said that the patterns are not that useful for their work because they are mostly old stuff.

mapped into a specific programming environment to be used.

It also means that often we can look at daily life to help design successful messaging solutions.

How about more complex conversations, transactions etc?