

**founder & ceo**

Fuat Kircaali fuat@sys-con.com

**group publisher**

Roger Strukhoff roger@sys-con.com

**advertising**

**senior vp, sales & marketing**

Carmen Gonzalez carmen@sys-con.com

**advertising sales director**

Megan Mussa megan@sys-con.com

**associate sales manager**

Corinna Melcon corinna@sys-con.com

**sys-con events**

**president, events**

Carmen Gonzalez carmen@sys-con.com

**events manager**

Lauren Orsi lauren@sys-con.com

**events associate**

Sharmonique Shade sharmonique@sys-con.com

**customer relations**

**circulation service coordinator**

Edna Earle Russell edna@sys-con.com  
Alicia Nolan alicia@sys-con.com

**sys-con.com**

**vp, information systems**

Bruno Decaudin bruno@sys-con.com

**information systems consultant**

Robert Diamond robert@sys-con.com

**web designers**

Stephen Kilmurray stephen@sys-con.com  
Richard Walter richard@sys-con.com

**accounting**

**financial analyst**

Joan LaRose joan@sys-con.com

**accounts payable**

Betty White betty@sys-con.com

**subscriptions**

Subscribe@sys-con.com

Call 1-888-303-5282

For subscriptions and requests for bulk orders, please send your letters to Subscription Department

Cover Price: \$8.99/issue

Domestic: \$89.99/yr (12 issues)

Canada/Mexico: \$99.99/yr

All other countries \$129.99/yr

(U.S. Banks or Money Orders)

Back issues: \$12 U.S. \$15 all others

# Object-Oriented Pizza

## Procedural programming and OO programming buzzwords



By Jeffry Houser

Over the past few years

it's gotten a lot harder

to write ColdFusion

code. That's not to say

that CFML has changed significantly. You can

still write code today that's much the same as

what you wrote five years ago or longer.

However, with the introduction of CFCs in CFMX, ColdFusion started to feel like a real programming language for the first time. CFCs, as you probably know, let us encapsulate data and functionality. With this new tool in the toolbox of CF developers, many slowly started applying advanced programming concepts to their development. This caught on and things have spiraled.

Unfortunately, many CF developers come from a non-programming background. There is all this focus on object orientation, design patterns, inheritance, interfaces, polymorphism, and (insert your own buzzword here) can be quite daunting. Traditional "procedural" development is often considered to be the one true evil. It can be



hard to discover where the real value lies.

Applying all this "stuff" to your development often results in longer upfront development time. In theory you'll gain it back in ease of maintenance. Unless you're a better developer than I, the reality is that your first OO attempts will turn into a spaghetti code mess. It may even be worse than many old CF spaghetti code applications, since you're adding a layer of complexity on top of the simple approach. In this article, I'm going to help you try to make sense of the buzz, and give a comparison of procedural approaches and OO approaches along the way.

### Create Your First Pizza

What happens when you order a pizza? It's probably something like this:

- Customer comes into a restaurant.
- Customer places order with the waitress.
- Waitress gives order to the cook.
- The cook prepares the pizza and puts it in the oven.
- When the pizza is ready, the cook takes the pizza out of the oven and cuts it.
- The waitress picks up the pizza and delivers it to the customer.

This is a generic enough algorithm. There may be differences and complications depending on a variety of factors, but for the purposes of this article, we can assume that we're building a system to follow that algorithm.

Let's pretend that our pizza restaurant is futuristic, and all the workers are robots. It's our task to write a program that sends out instructions to the waitress robot and cook robot so that they can operate the pizza restaurant. If you were to implement this as a traditional system for processing orders, you might come up with something like this:

- Wait for new customer.
- When customer shows up, waitress gets order from customer.
- Waitress sends order to the cook.

- The cook prepares the dough, add sauce, cheese, and relevant toppings.
- The cook puts the pizza in the oven to bake.
- Wait until pizza is ready.
- Once ready, the cook takes the pizza out of the oven .
- The cook cuts the pizza cuts the pizza.
- Cook gives pizza to the waitress.
- Waitress gives pizza to the customer.
- Go back to the beginning and wait for a new customer...

This algorithm will definitely get the job done and tell our robots how to deal with the orders. For purposes of this example, we're going to assume that our system is so efficient at processing orders that we don't have to worry about waiting on multiple customers at once. Or if you don't buy that, pretend business is really bad.

## Elements of the System

Based on the descriptions of the algorithm and the first programming approach, can you identify a few of the elements in this system? Here are a few of the important elements I noticed:

- **Customer:** The customer is the guy who wants the pizza. He doesn't do anything except provide the order to the waitress robot.
- **Waitress:** This is the robot that takes your order.
- **Pizza:** The pizza is an element of the system, with cheese, sauce, and all the various toppings, especially pepperoni.
- **Cook:** This is the robot that prepares and cooks your order.

These elements come into play as we explore different implementation options for this system.

## The First Implementation Attempt

Back in the dark ages of CF development, you might have implemented the system like this:

```
<cfif IsDefined("newOrder")>
  <cf_PreparePizza orderdetails=" newOrder" returnVariable="newPizza">
  <cf_BakePizza pizza=" newPizza" returnVariable=" newPizza">
  <cf_CutPizza pizza=" newPizza" returnVariable=" newPizza">
  <cf_DeliverPizza pizza=" newPizza">
</cfif>
```

Our initial algorithm had a "wait until order" clause, which doesn't apply very well to CF development. So, this code assumes that every order is provided as a page-submit, and we process it on that page. Prior to CFCs, the functionality could only be encapsulated via custom tags or UDFs. This

example uses custom tags for preparing the pizza, baking the pizza, cutting the pizza, and delivering the pizza.

With this approach, functionality is encapsulated into custom tags. Data isn't encapsulated. We have two data structures in our main template, newOrder and newPizza. These data structures are passed in and out of the custom tags. This is a procedural approach to the problem. We broke things up into separate chunks, and are just telling the



Figure 1: The pizza store object model

computer the order with which to run our code. I'm going to leave the implementation details of the data structures and custom tags to your imagination. I haven't had a client ask me to control robots from a ColdFusion program yet.

## So, Let's Put Some CFCs into the Mix

This whole example is about pizza. Most of the functions we're doing are being done on the pizza. So, we can create a Pizza CFC to handle a lot of this functionality. A few methods to put in the CFC are Bake, Prepare, Cut, and Deliver. Instance variables in the CFC might be price, ingredients, and the order information. Instead of executing multiple custom tags and passing data back and forth, we pass the data into the pizza component, and then merely call the component's methods (or functions) to act on the data. A sample of this approach can be seen in Listing 1. As with the custom tags, I didn't flesh out the code of the CFC.

Is the CFC in this example an object? Technically it meets the definition of an object from the OO paradigm. It also meets the definition of an Abstract Data Type (ADTs) from the procedural paradigm. For all intents and purposes objects and ADTs are the same. They are containers for data that also abstract functionality to perform actions on the data. Today everyone calls them objects because that's a buzzier word.

## Add Some Object Orientation

Object-oriented programming is supposed to build a model that mimics the way things really operate in the real world. In procedural programming, your model is built to make it easy for the computer to process. I once heard it said that procedural is requesting the data and doing things to it. Using this as a basis, Listing 2 is very procedural. It's not mimicking the real world; it's breaking things up into small chunks to send to the computer processor for processing.

A pizza doesn't know how to deliver itself to the customer. It doesn't know how to prepare itself. It does not know how to bake itself, or cut itself into slices. In an OO model, this functionality wouldn't be put on a pizza object. An OO is geared towards telling your "things" how to act on the data they already have. Next we'll modify the example to be a bit more OO.

The pizza doesn't actually do anything in the real world. The robots or customers do the real tasks. If we were to create an object for it, it would just be a data container to pass around to objects. Here are a few of the actions that occur in the system:

- **OrderPizza:** This would be something that the customer does.
- **PreparePizza:** This is something that the cook does.
- **BakePizza:** This is also something done by the cook.
- **Cut Pizza:** The cook cuts the pizza.
- **Receive Order:** When the customer orders the pizza, the waitress gets the order. When the waitress gets the order, she needs to pass it to the cook. As such both the waitress and the cook may have a method for receiving the order.
- **Receive Pizza:** When the cook is done cutting the pizza, he needs to give it to the waitress. The waitress has to respond to the receive pizza method. When the waitress gets the pizza, it must be delivered to the customer, and as such the customer must also respond to the Receive Pizza method.

We come out of this description with a customer object, a cook object, and a waitress object. You can see these in Figure 1. Most likely a full-fledged application would also include a pizza object and an order object, used as data containers as arguments to the methods.

With all these objects, our code changes quite a bit. This line will start the order process:

```
Customer.OrderPizza();
```

This assumes that a customer object is already created, of course. The customer OrderPizza function would look like this:

```
<cffunction name="OrderPizza">
  <cfset Waitress.RecieveOrder(MyOrder)>
</cffunction>
```

The Waitresses ReceiveOrder method would look like this:

```
<cffunction name="OrderPizza">
  <cfargument name="Order" type="order">
  <cfset Cook.RecieveOrder(MyOrder)>
</cffunction>
```

The cook's ReceiveOrder method would look like Listing 2. The cook does a lot of work in this process compared to the customer or waitress. The waitress gets the pizza back from the cook:

```
<cffunction name=" RecievePizza">
  <cfargument name="pizza" type="Pizza">
  <cfset customer. RecievePizza (arguments.pizza)>
</cffunction>
```

The customer gets the pizza from the waitress using this method:

```
<cffunction name=" RecievePizza">
  <cfargument name="pizza" type="Pizza">
  <cfset customer. Devour (arguments.pizza)>
</cffunction>
```

Instead of one CFC with lots of data and functionality, we have a lot of CFCs with very little data and functionality. The order is passed around, down the chain from the customer to the waitress to the cook. The cook processes it and sends a pizza back to the customer.

## Conclusion

A lot of ColdFusion programmers started programming without using any encapsulation at all. Business logic and display code was mixed in a single template. This was partially due to limitations within the language of ColdFusion. When I talk to them they refer to this old school CF method of programming as procedural, even though it had very little to do with procedural programming. I hope this article enlightened you to a few of the differences between procedural programming and object-oriented programming.

## About the Author

*Jeffrey Houser has been working with computers for over 20 years and in Web development for over 8 years. He owns a consulting company and has authored three separate books on ColdFusion, most recently ColdFusion MX: The Complete Reference (McGraw-Hill Osborne Media).*

[jeff@instantcoldfusion.com](mailto:jeff@instantcoldfusion.com)

### Listing 1

```
<cff IsDefined("newOrder")>
<cfscript>
  newPizza = Createobject('component', 'pizza');
  newPizza.init(newOrder);
  newPizza.prepare();
  newPizza.Bake();
  newPizza.cut();
  newpizza.deliver();
</cfscript>
```

### Listing 2

```
<cffunction name="OrderPizza">
  <cfargument name="Order" type="order">
  <cfset var pizza = CreateObject('component', 'pizza')>
  <cfset variables.order = arguments.order>
  <cfset PreparePizza(pizza)>
  <cfset BakePizza(pizza)>
  <cfset CutPizza(pizza)>
  <cfset Waitress.RecievePizza(pizza)>
</cffunction>
```

Download the Code...  
Go to <http://coldfusion.sys-con.com>