ESB 101

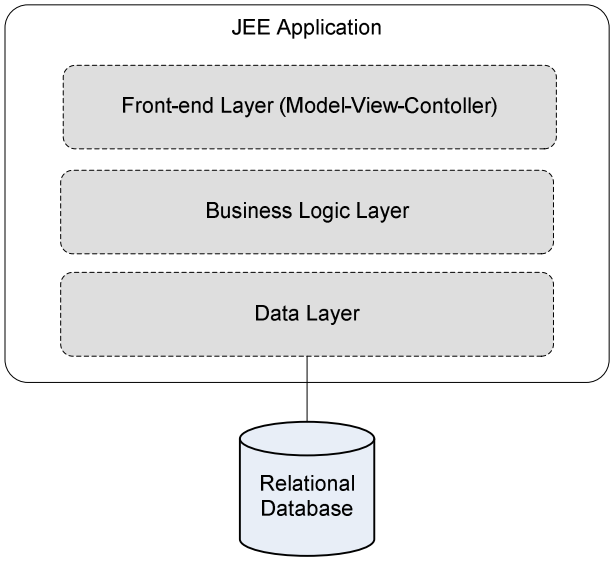
Here is a primer.

In the beginning, there was the bit. It could do two things, store a 1 or store a 0. That was it. It is really the building block of everything we do. After it was built the inventor thought about it and said "Hmm…now how can I get things to communicate with this?" Fast forward sixty years and the ESB was born!

That really is the heart of the ESB. Getting bits from location #1 to location(s) 2,3,..N. But wait, can't other things get bits from location 1 to location N? The answer is YES! This is where we get to discuss the evolution of architecture.

N-Tier

For brevity I'll skip ahead to the birth of n-tier architecture. Here is what it looks like in a diagram form.



Note that in this case there are three tiers, a data tier, a logic tier and a presentation tier. So this shows a 3-tier architecture. It is quite common for an application to have this sort of architecture.

The business users are happy, the developers are happy, and all is well in the world.

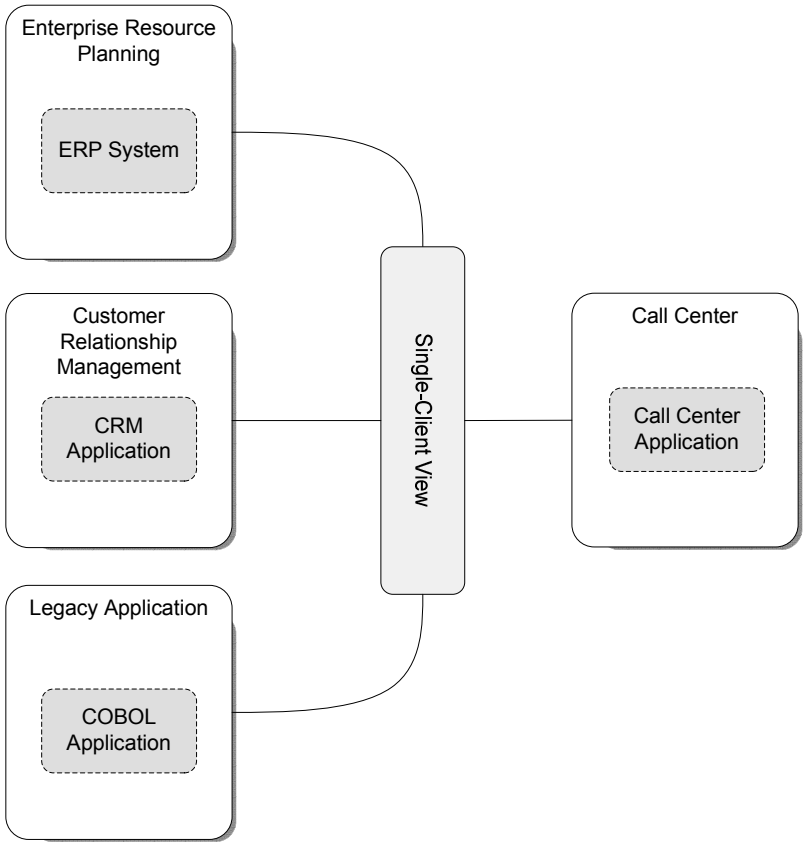
Well, that was true, until there were TWO of these applications. Now the business users started to grumble because they have to use two applications to get their jobs done. Then a third app is created, then a fourth, and so on and so on.

Developers around the world came together and came up with a solution. Let's have ONE presentation tier, and have it pull the data from the different applications. Thus the portal was born.

Author: Aaron Korver

Portal

The portal consists of the Single-Client View that ties to the different applications. So the n-tier still exists in each of the application, but now the business gets a single view to see the data they need to get their job done.



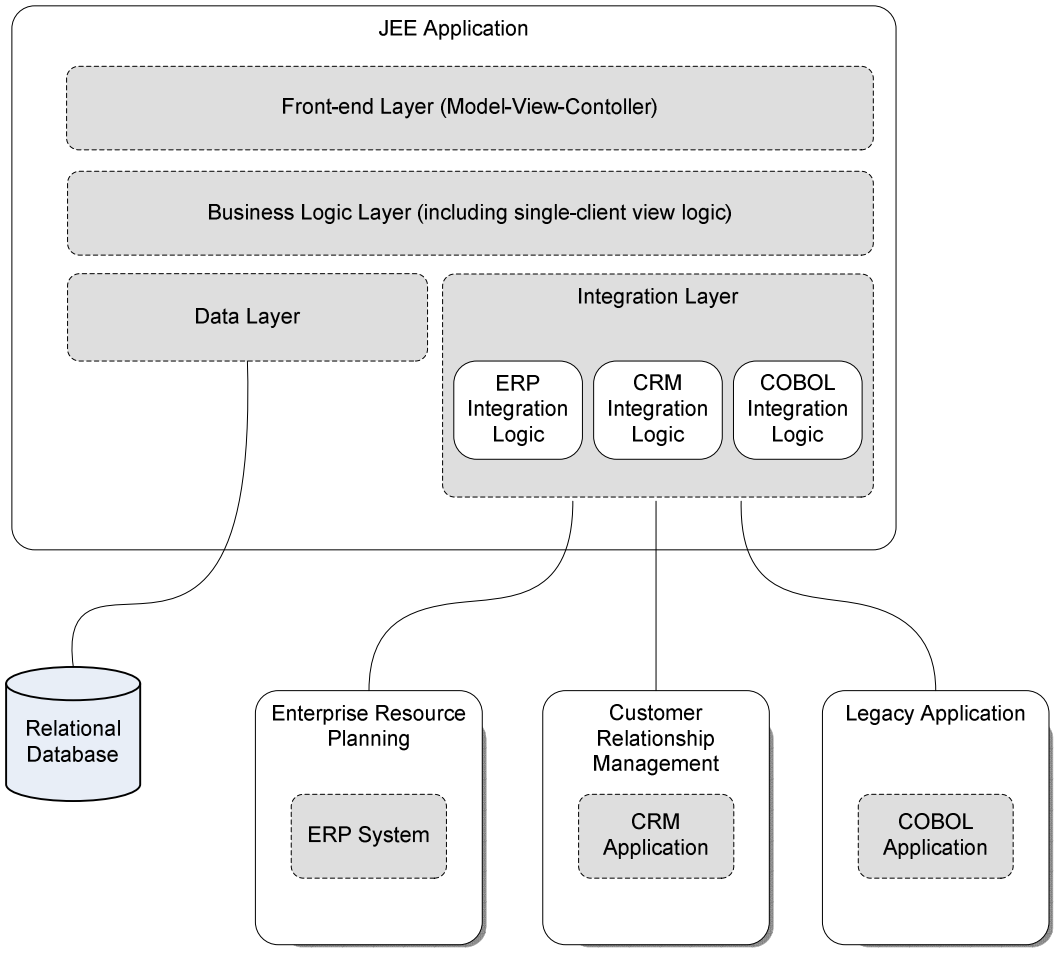
Each team can be responsible for managing and enhancing their application. The piece that each app must provide to the portal is a presentation.

It is important to note here that in order to have everything play nice, there MUST be a standard API that everyone agrees with. Without this, the portal architecture would not work.

The business was happy, the developers were happy, and all was well in the world.

Now the company looks at the portal and realizes, "Hey! We have lots of duplication across these N applications." This was quite true. Since each was written individually, all the shared logic is duplicated in the N-business layers. Also, the apps share the same data! So we have duplication in the N-data layers. The last problem here is that these individual applications don't really live independently. They all have to work together, they have to share information, and share responsibilities. How can we solve these problems? Time for a REWRITE!

The Integration Application



One way to architect the rewrite to reduce duplication across the enterprise is shown above. What we have done is to create another application! We'll call this the Integration Application. We still have n-tier architecture in place. And we also still have the original applications in place.

What we have added here is a unified business layer. This layer holds the shared business logic as well as logic needed for the portal view. The really important part of this new app is the Integration Layer. All of the code to get the applications to talk to each other is placed in this layer. Also, the code to talk to the individual applications is placed here.

IMPORTANT!! There is a hidden thing in the diagram above. The Integration Layer has to talk to the individual applications. HOW? It doesn't use the presentation layer, so the only option we have is to somehow expose the guts of these applications. So the old apps get a new layer in their architecture called the Service Layer.

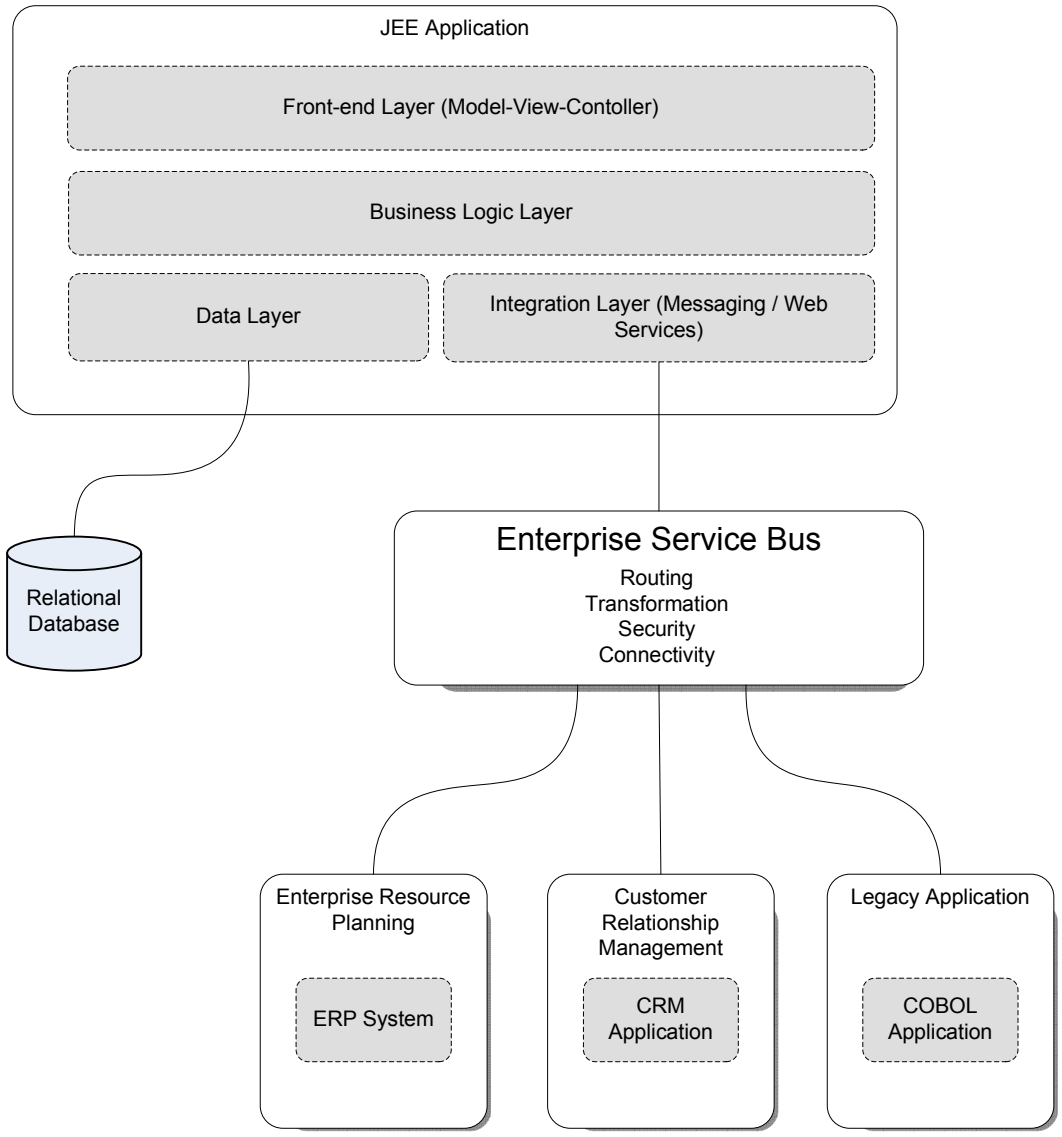
We have reduced duplication, and consolidated functionality into a nice neat package. All is good right?

The Problem (and maybe the solution?)

All is good until things get messy. A common theme here is COMPLEXITY. See, all systems start out simple, and then grow, and grow, and grow. Pretty soon, we are either left with the original application + bolted on enhancements, or we face the complexity head on and update the system.

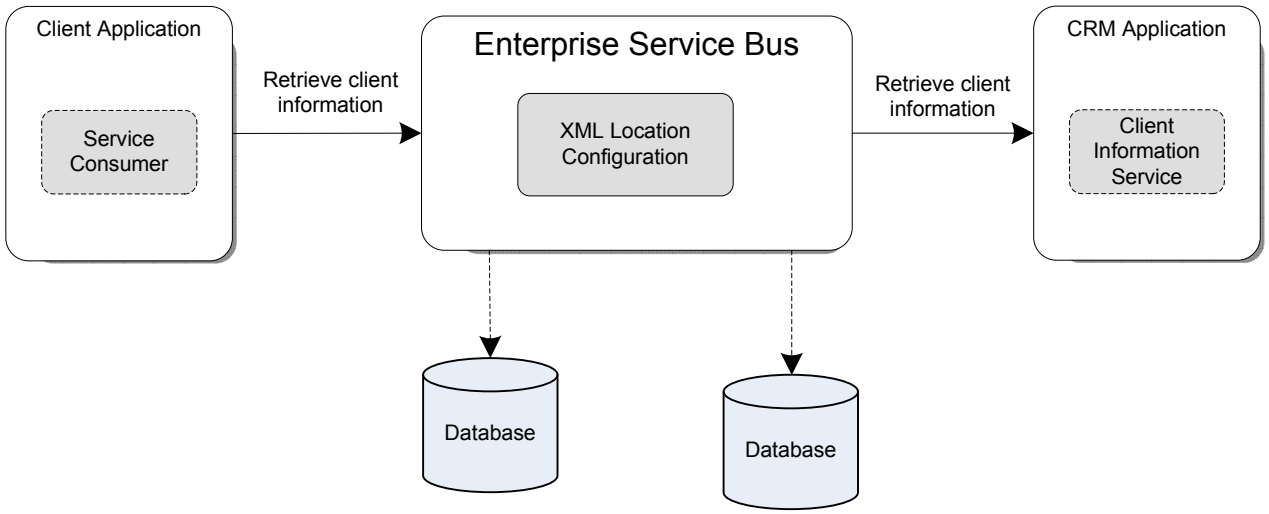
As more applications get added to our Integration Application, the Integration Layer gets more and more complex. There are more connections being made, more resources to manage. So what is a developer to do?

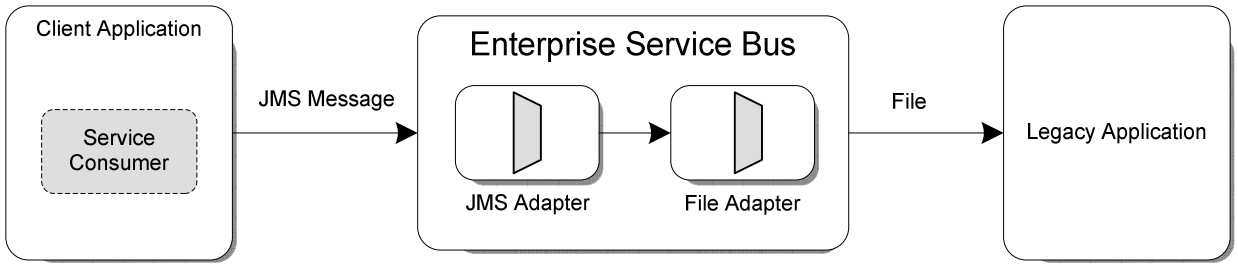
One possible solution is in the diagram below. Same Integration Application, but now we have decoupled the logic in the Integration Layer from all the other stuff. This "other stuff" has been moved into what is commonly called the Enterprise Service Bus (ESB). And that is how the ESB came about.

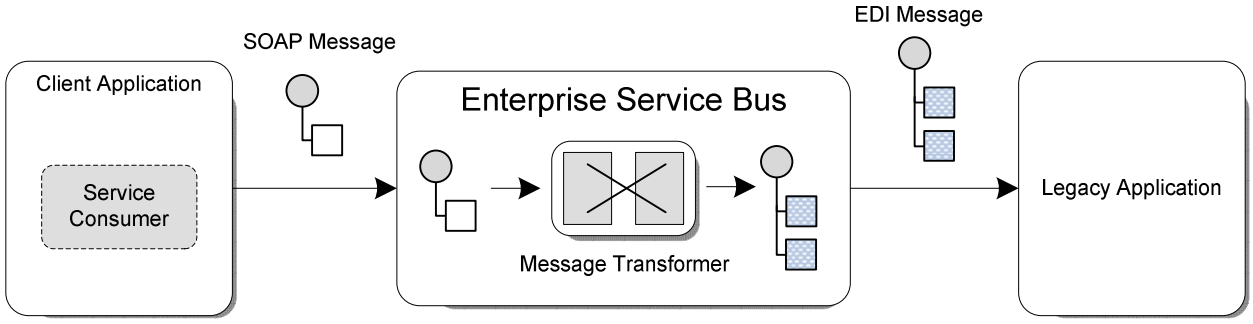


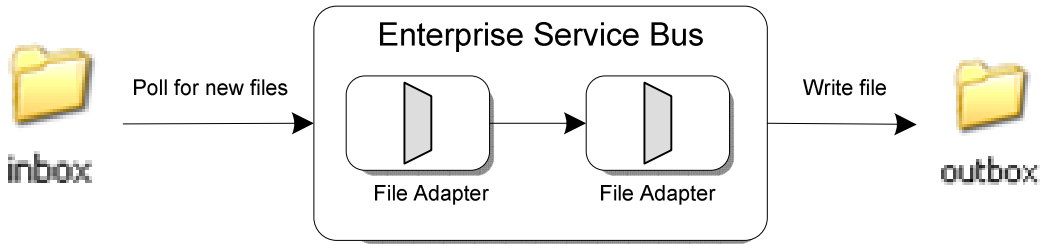
The Swiss Pocket Knife of Architecture

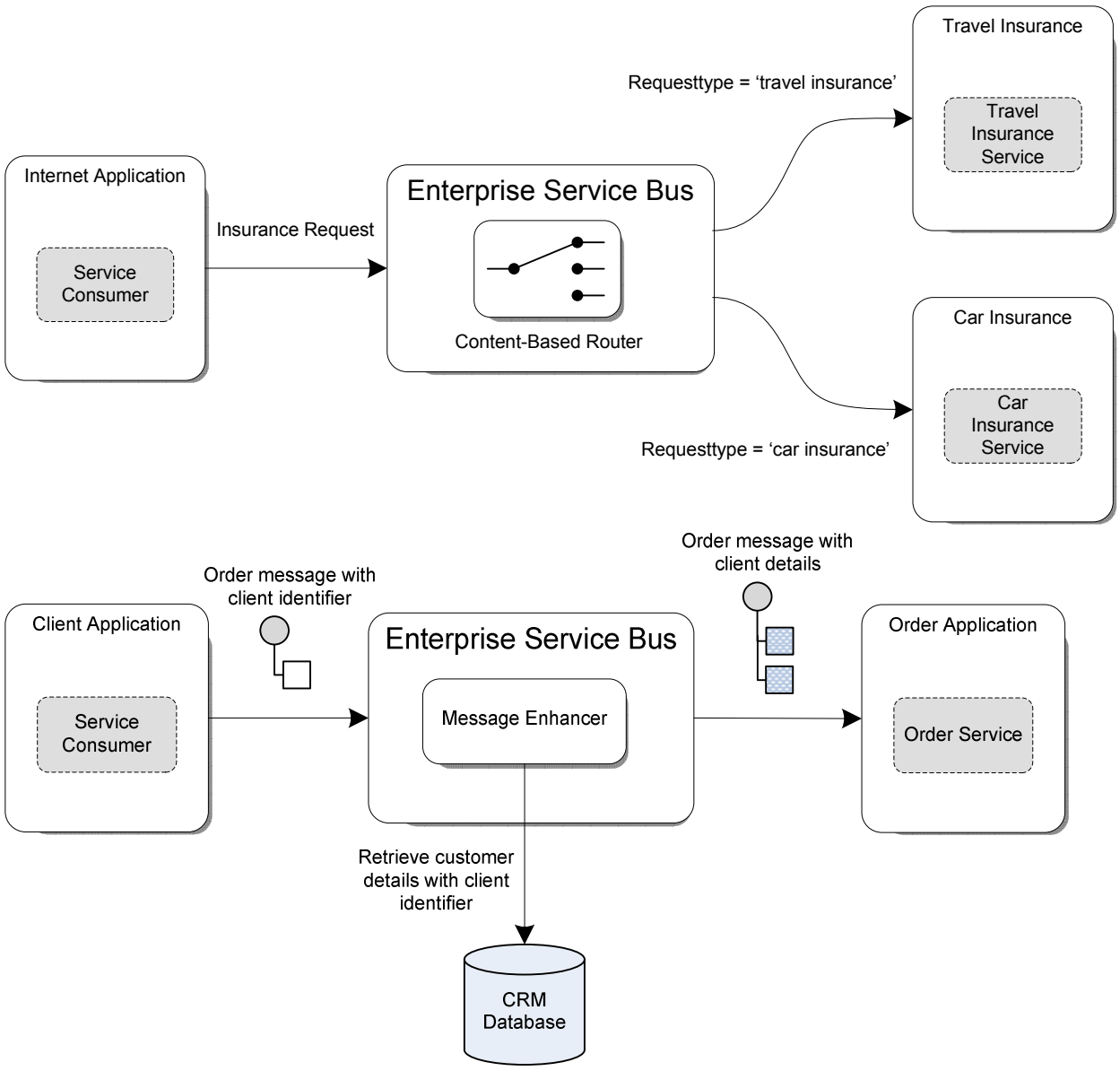
Let us take a look at how we could use an ESB by giving some examples.

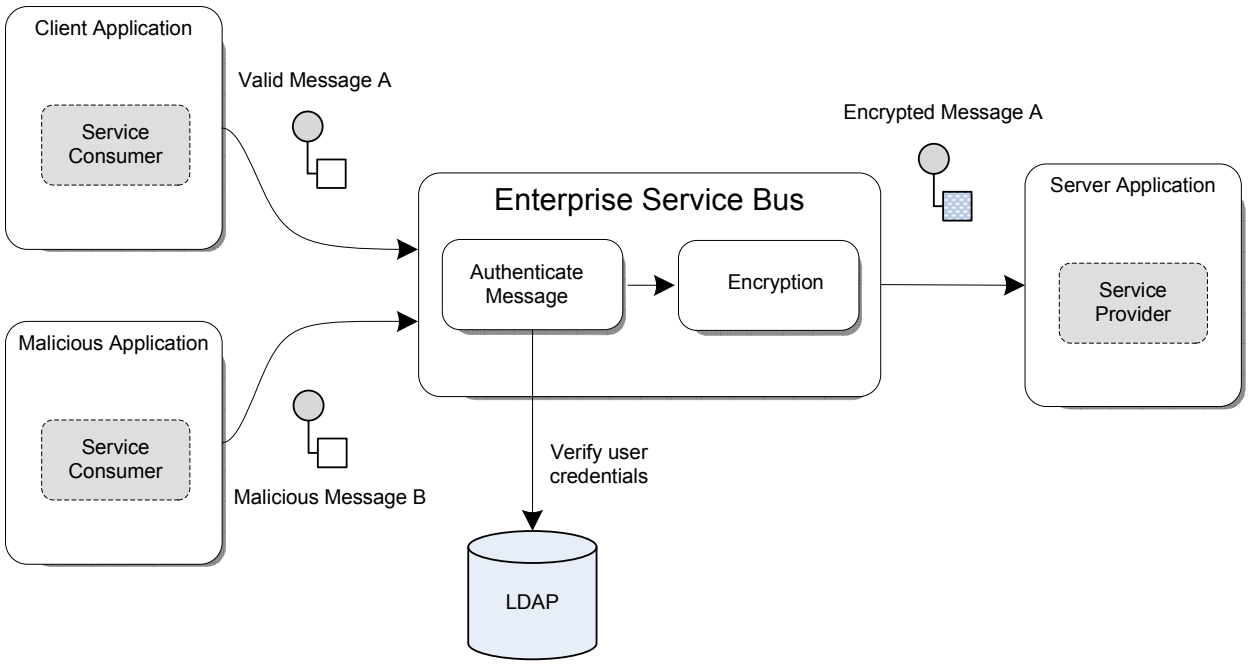









Travel Insurance
Requesttype = 'travel insurance'
Travel Insurance Service
Internet Application
Service Consumer
Insurance Request
Enterprise Service Bus
Content-Based Router
Car Insurance
Car Insurance Service
Requesttype = 'car insurance'
Order message with client details
Order message with client identifier
Client Application
Service Consumer
Enterprise Service Bus
Message Enhancer
Order Application
Order Service
Retrieve customer details with client identifier
CRM Database


Client Application
Service Consumer
Valid Message A
Encrypted Message A
Enterprise Service Bus
Authenticate Message
Encryption
Server Application
Service Provider
Malicious Application
Service Consumer
Malicious Message B
Verify user credentials
LDAP