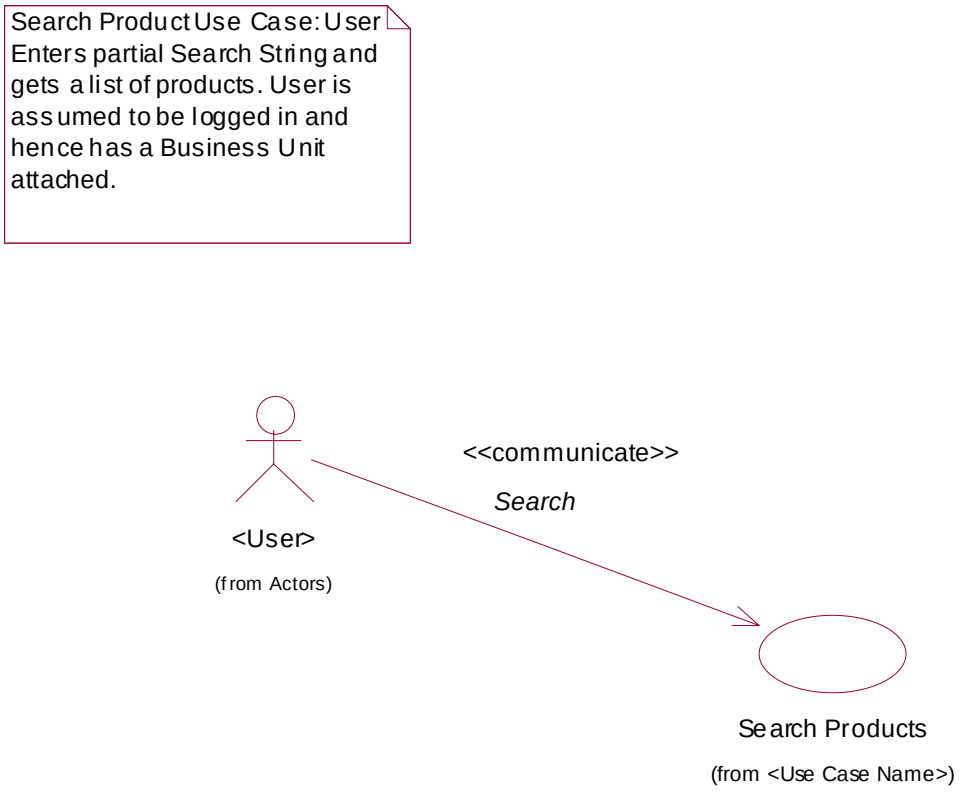
RTPDEMO – Architecture and high-level design

Use Case under consideration:



This is my attempt to describe the architecture - I have in mind. I assume you have read the specs / requirement document.

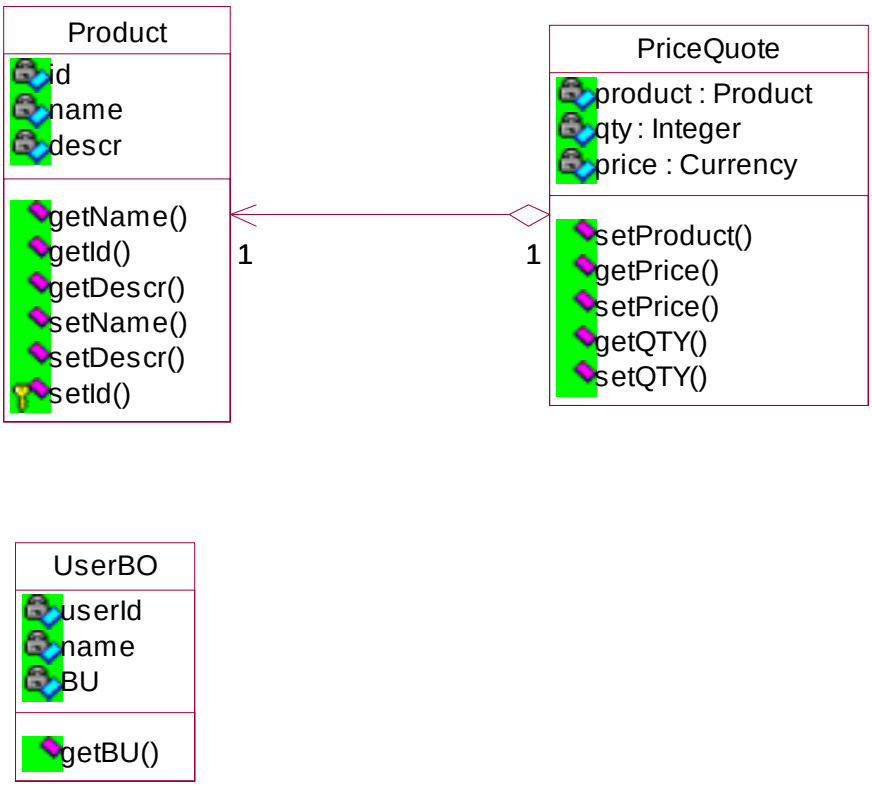
A few core assumptions:

1> There are a few back end systems (systems of record) already built with different technologies. This App will just provide a 'composite' view over the underlying systems.
2> For simplicity (and other reasons we will discuss later) the app does not use any EJBs. One of the systems that we will integrate may have been built using EJB. But our app as such will not use EJB – to start with (please go with this for now, I know saying 'we will not use EJB', to J2EE architects, is a blunder. But – please for now!)

OK. Now, as described in the above Use Case diagram the user searches for products – the system returns a list of products.

A preliminary domain model is defined as:

The domain Model:



The PriceQuote contains reference to the Product.
User has an attribute – BU that may have values : WS / JDO / EJB & so on …

Not spending too much time on the domain model, lets go ahead.

Hopefully by now, you have realized the KEY difference between your traditional J2EE App – object model and our application – object model. Not so much in the model as such but in terms of: where do the objects themselves belong?

In traditional application, Objects are bound to one & only one persistent store.

In our application depending on the BU (that maps to the back end system) Objects are mapped to different persistent stores.

Hence there is no single OR mapping needed. Rather for a relational store we need a OR mapping, for say a web service we need to map Objects to XML (WSDL or Schema).

OK, just to summarize:

1> Objects are mapped to different back end systems.
2> As far as possible we should keep this transparent to other layers of the system (view / controller)

Realization of Search Products Use Case:

So we have defined the Product. PriceQuote & UserVO Objects. These objects s will be transferred back & forth, between the various layers of the application.

As we discussed these objects are persisted in various systems. But the STRUTS layer should be transparent to it. For example it should just send the search string and get a list of Product Objects without knowing about where the products are actually coming from.

This is done using our 'adapter' layer. The Real Time Pricing application defines one adapter per back end system it integrates to.

The Adapter factory takes the USER Object as input and based on the BU value in the USER object, it will hand you the right adapter.

All the Adapters provide the following APIs:

```
Public interface ProductPriceIF{

// GetProducts:
Public ArrayList getProducts (String nameSearchString);

//GetPrice:

Public PriceQuoteVO getPrice (ProductVO pv, int qty);
}
```

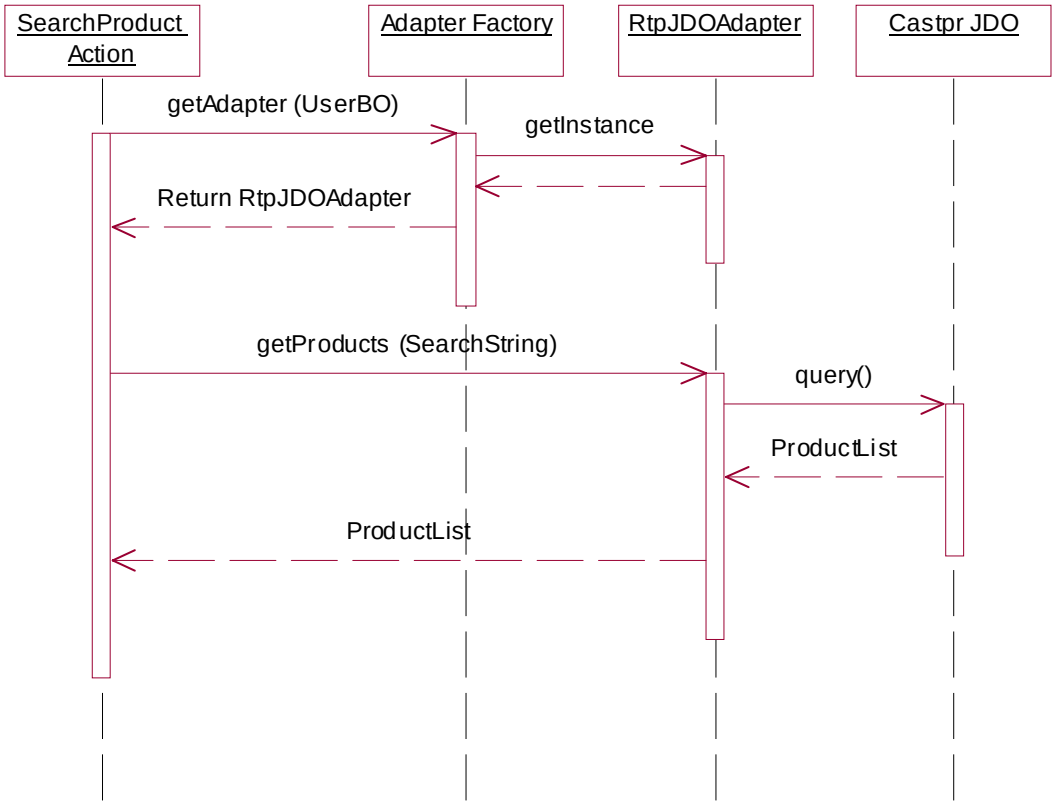
In other words the adapters adapt this ProductPriceIF interface to the system they are integrating with.

The JDO adapter connects to a relational store and selects the product table and converts the o/p to a List of PRODUCTobjects and so on..

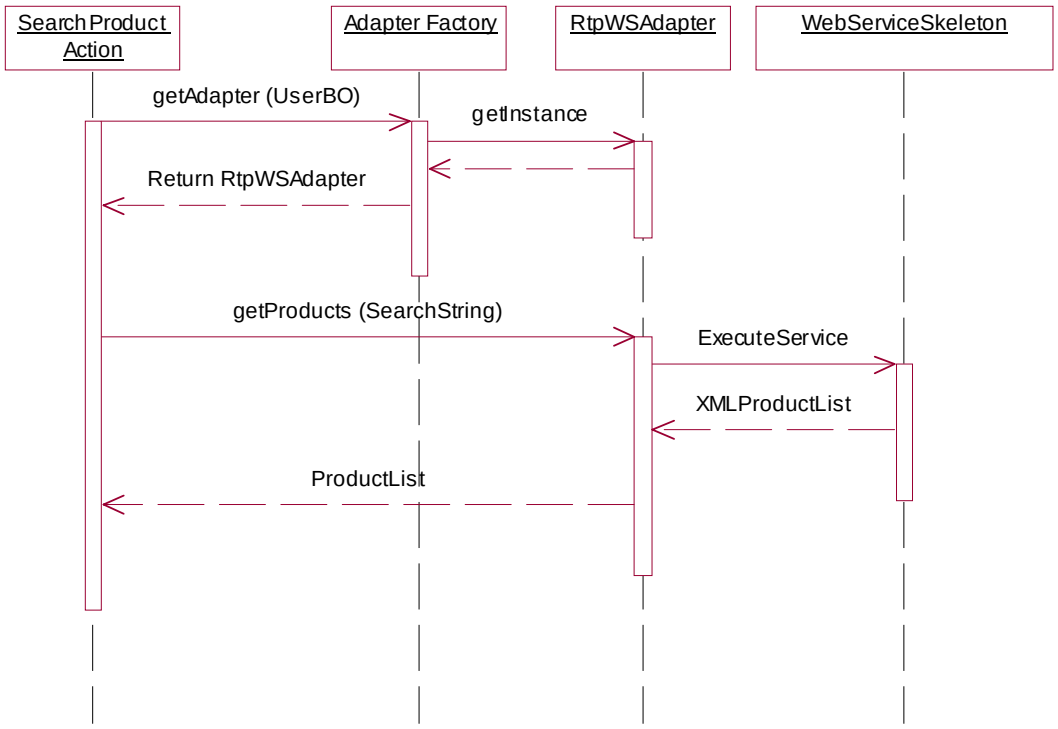
The Web Service Adapter uses the Skeleton classes generated to ‘consume’ or ‘execute’ the Service that returns product list document. It then converts it into a list of PRODUCT objects.

This is shown in the following sequence diagrams:

1> When UserBO.BU = JDO:



2> Use Case realization when BU = WS (Web Service)



Similarly other adapters 'adapt' the ProductPriceIF to respective back end systems.

NOTE: We could call these 'Facades' instead of adapters. Depends on the way you look at it. But I think Adapter 'word' is easily understood in Enterprise IT world (thanks to the EAI vendors). This exact same pattern is used by many EAI tools.

EJB or Not EJB

One could just use a session façade instead of the adapter layer. I have purposely tried to avoid the use of EJBs. For two reasons:

1> As per the requirements we do not have a lot of business logic/ persistence and transaction requirements. This is a classic case in many Real Time Enterprise Applications. The systems of records (back end systems) manage the core business process and business logic. But those are more 'departmental' focused. The RTE – applications – are built on top of these system of records and provide a thin layer of business process flow across these systems. Please look at the 'Architect' section on the www.theopenstack.com for more details.
2> For the data access part – where we need to search products from the data base , we could use EJB layer. Here my thought is to demonstrate how to start with a simple JDO layer (CASTOR JDO is way simpler compared to EJBs – for a new J2EE developer – more like VB!). Obviously the simpler technology has its

limitations. But we can demonstrate how easily we can add an EJB layer if needed. (in fact we are in process of doing so!)

Anyway, with the current architecture, an Enterprise developer could build a RTE application using just STRUTS & Tomcat (for deployment)!