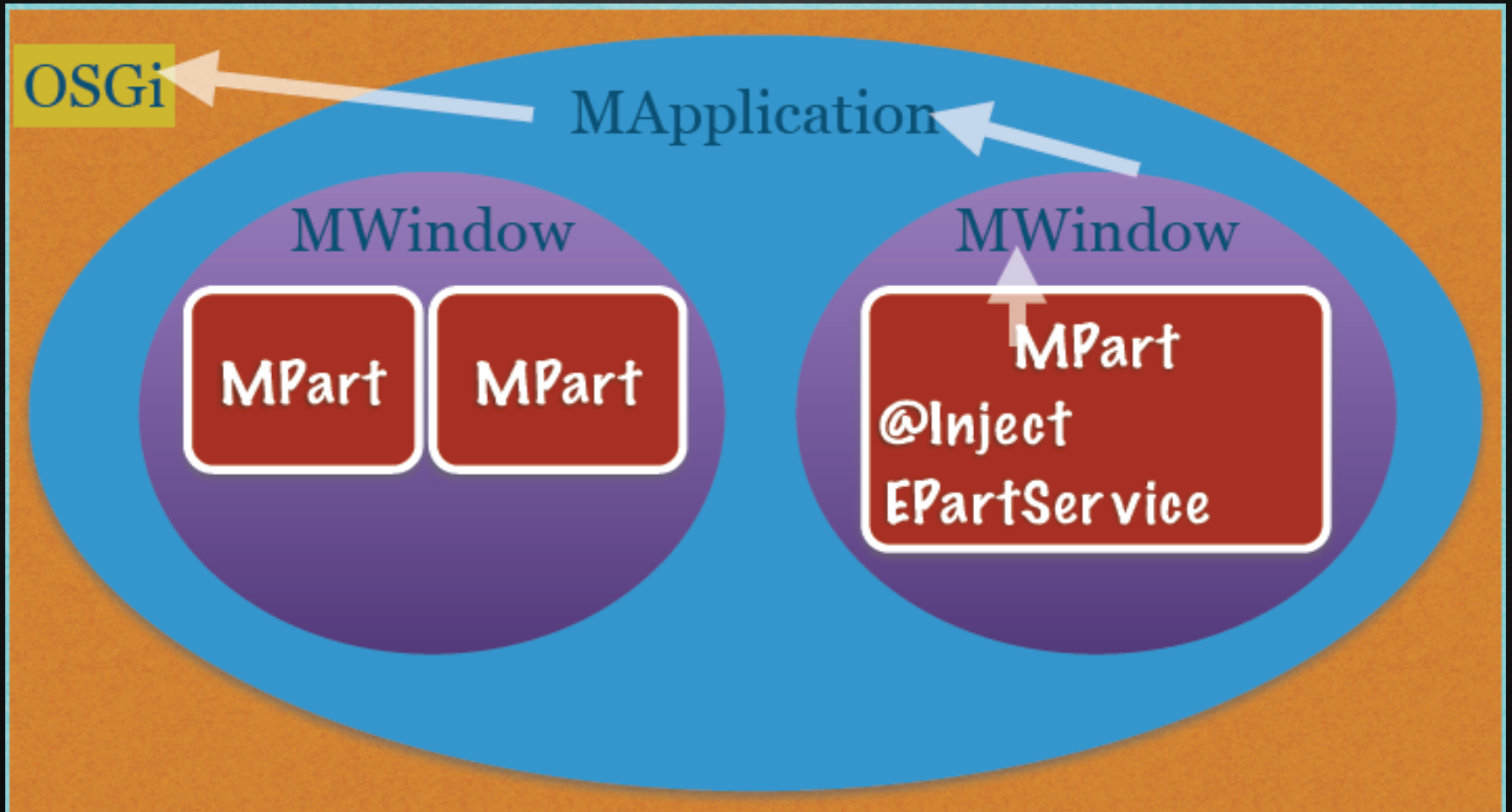


CONTEXT FUNCTIONS

CLASSIC CONTEXT LOOKUP



CONTEXT FUNCTION

- goes in when a specific **key** is requested
 - EPartService
 - EHandlerService
 - EYourCoolService, EYourCoolObject
- gives you the **context** in which this key was requested
 - Part 1 context?
 - Window 2 context?
- you implement the F in $F(\text{key}, \text{context})$
 - so that it returns the runtime object

GUTS OF THE PLATFORM

@Inject private EHandlerService
handlerService

```
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0" name="org.eclipse.e4.core.handlers">  
  <implementation class="org.eclipse.e4.core.commands.internal.HandlerServiceCreationFunction"/>  
  <service>  
    <provide interface="org.eclipse.e4.core.contexts.IContextFunction"/>  
  </service>  
  <property name="service.context.key" type="String" value="org.eclipse.e4.core.commands.EHandlerService"/>  
</scr:component>
```

```
public class HandlerServiceCreationFunction extends ContextFunction {  
    @Override  
    public Object compute(IEclipseContext context) {  
        return ContextInjectionFactory.make(HandlerServiceImpl.class, context);  
    }  
}
```

```

public Object compute(IEclipseContext context) {
    // look for the top-most MWindow in the context chain:

    // 1st: go up the tree to find topmost MWindow
    MWindow window = null;
    IEclipseContext current = context;
    do {
        MContext model = current.get(MContext.class);
        if (model instanceof MWindow)
            window = (MWindow) model;
        current = current.getParent();
    } while (current != null);

    if (window == null) {
        if (context.get(MApplication.class) != null) {
            // called from Application scope
            return ContextInjectionFactory.make(ApplicationPartServiceImpl.class, context);
        }
        return IInjector.NOT_A_VALUE;
    }

    IEclipseContext windowContext = window.getContext();
    PartServiceImpl service = windowContext.getLocal(PartServiceImpl.class);
    if (service == null) {
        service = ContextInjectionFactory.make(PartServiceImpl.class, windowContext);
        windowContext.set(PartServiceImpl.class, service);
    }
    return service;
}

```

@Inject
EPartService

OBJECT SUPPLIERS

What an injector needs

```
public interface IInjector {  
    * Methods may return this to indicate that the requested object was not found..  
    final public static Object NOT_A_VALUE = new Object();  
  
    * Injects data from the supplier into a domain object. See the class comment for  
    public void inject(Object object, PrimaryObjectSupplier objectSupplier) throws I  
  
    * Un-injects data from a domain object into the supplier.  
    public void uninject(Object object, PrimaryObjectSupplier objectSupplier)  
  
    * Call the supplier to inject data into the object.  
    public Object inject(Object object, PrimaryObjectSupplier objectSupplier)  
  
    * Call the supplier to uninject data from the object.  
    public Object uninject(Object object, PrimaryObjectSupplier objectSupplier)  
  
    * Call the supplier to inject data into the object.  
    public Object invoke(Object object, Class<? extends Annotation> qualifier, Object
```

Type hierarchy of 'org.eclipse.e4.core.di.suppliers.PrimaryObjectSupplier':

- Object - java.lang
 - PrimaryObjectSupplier - org.eclipse.e4.core.di.suppliers
 - ContextObjectSupplier - org.eclipse.e4.core.internal.contexts

Press 'Ctrl+T' to see the supertype hierarchy

Extended Object Supplier

- goes in when an object is annotated with a specific annotation
 - @UIEventTopic Object object;
 - @UserData("admin") User user;
- gives you all you need (requestor, qualifier data etc.)
- needs you to describe how the injection of the dependency is made

SPECIFICALLY

- Link with the annotation

```
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0" name="os.demo.suppliers">
  <implementation class="os.demo.suppliers.UserDataObjectSupplier"/>
  <service>
    <provide interface="org.eclipse.e4.core.di.suppliers.ExtendedObjectSupplier"/>
  </service>
  <property name="dependency.injection.annotation" type="String" value="os.demo.suppliers.annotation.UserData"/>
</scr:component>
```

- Implement the DI logic

```
public class UserDataObjectSupplier extends ExtendedObjectSupplier{

    @Override
    public Object get(IObjectDescriptor descriptor, IRequestor requestor,
        boolean track, boolean group) {

        return null;
    }
}
```

Meanwhile in the client of the DI

```
@PostConstruct  
public void createComposite(Composite parent, @UserData User user){  
    parent.setLayout(new GridLayout());  
}
```

