# StatefulServiceModel

A lot of the code quickly stolen from PooledServiceModel - therefore some suboptimal namings

```
package scm.hivemind.statefulservice;

import org.apache.hivemind.ApplicationRuntimeException;
import org.apache.hivemind.HiveMind;
import org.apache.hivemind.ShutdownCoordinator;
import org.apache.hivemind.events.RegistryShutdownListener;
import org.apache.hivemind.impl.ConstructableServicePoint;
import org.apache.hivemind.impl.ProxyUtils;
import org.apache.hivemind.impl.servicemodel.AbstractServiceModelImpl;
import org.apache.hivemind.internal.Module;
import org.apache.hivemind.service.ThreadCleanupListener;
import org.apache.hivemind.service.ThreadEventNotifier;


/**
 * A Hivemind service model for services with (long-lasting,
 * i.e. request-spanning) state.
 *
 * implementation is currently based on pooled service model.
 *
 * @author Marcus Schulte
 */
public class StatefulServiceModel extends AbstractServiceModelImpl {

        /**
     * Name of a method in the deferred proxy that is used to obtain the constructed service.
     */
    protected static final String SERVICE_ACCESSOR_METHOD_NAME = "_service";

    private Object _serviceProxy;

    private ThreadEventNotifier _notifier;

    private ThreadLocal _activeService;

    private ClientStateStorage _stateStorage;

    /** @since 1.1 */

    private Class _serviceInterface;

    /**
     * Shared, null implementation of StatefulServiceLifecycleListener.
     */
    protected static final StatefulServiceLifecycleListener NULL_MANAGEABLE
                                    = new StatefulServiceLifecycleListener()
    {
        public void resumeConversation()
        {
        }

        public void pauseConversation()
        {
        }

        public void terminateConversation() {}
    };

    private class StatefulService implements ThreadCleanupListener,

StateStorageClearanceListener
    {
        private Object _core;

        private StatefulServiceLifecycleListener _managed;
```

```
        /**
         * @param core
         *            the core service implementation, which may optionally implement
         *            {@link StatefulServiceLifecycleListener}
         */
        StatefulService(Object core )
        {

            _core = core;

            if (core instanceof StatefulServiceLifecycleListener)
                _managed = (StatefulServiceLifecycleListener) core;
            else
                _managed = NULL_MANAGEABLE;
        }

        public void threadDidCleanup()
        {
            unbindPooledServiceFromCurrentThread(this);
        }

        void activate()
        {
            _managed.resumeConversation();
        }

        void passivate()
        {
            _managed.pauseConversation();
        }

        public void clientStateCleared()
        {
            _managed.terminateConversation();
        }
        /**
         * Returns the configured service implementation.
         */
        public Object getService()
        {
            return _core;
        }

    }

    public StatefulServiceModel(ConstructableServicePoint servicePoint)
    {
        super(servicePoint);

        _serviceInterface = servicePoint.getServiceInterface();
    }

    public synchronized Object getService()
    {
        if (_notifier == null)
        {
            Module module = getServicePoint().getModule();

            _notifier = (ThreadEventNotifier) module.getService(
                    HiveMind.THREAD_EVENT_NOTIFIER_SERVICE,
                    ThreadEventNotifier.class);
        }
        if ( _stateStorage == null ) {
                        _stateStorage = (ClientStateStorage) getServicePoint().getModule()
                                                                              .getService(ClientStateStorage.
class);

                }
```

```
    if (_serviceProxy == null)
        _serviceProxy = constructServiceProxy();

    return _serviceProxy;
}

/**
 * Constructs the service proxy and returns it, wrapped in any interceptors.
 */
private Object constructServiceProxy()
{
    ConstructableServicePoint servicePoint = getServicePoint();

    if (_log.isDebugEnabled())
        _log.debug("Creating PooledProxy for service " + servicePoint.getExtensionPointId());

    Object proxy = ProxyUtils.createDelegatingProxy(
            "PooledProxy",
            this,
            "getServiceImplementationForCurrentThread",
            servicePoint);

    Object intercepted = addInterceptors(proxy);

    RegistryShutdownListener outerProxy = ProxyUtils
            .createOuterProxy(intercepted, servicePoint);

    ShutdownCoordinator coordinator = servicePoint.getShutdownCoordinator();

    coordinator.addRegistryShutdownListener(outerProxy);

    return outerProxy;
}

public synchronized Object getServiceImplementationForCurrentThread()
{
    if (_activeService == null)
        _activeService = new ThreadLocal();

    StatefulService pooled = (StatefulService) _activeService.get();

    if (pooled == null)
    {
        pooled = obtainPooledService();

        pooled.activate();

        _notifier.addThreadCleanupListener(pooled);
        _activeService.set(pooled);
    }

    return pooled.getService();
}

private StatefulService obtainPooledService()
{

    StatefulService result = getServiceFromClientStateStorage();

    if (result == null)
        result = constructStatefulService();

    return result;
}

private synchronized StatefulService getServiceFromClientStateStorage()
{
                return (StatefulService)
                    _stateStorage.retrieve( getServicePoint().getExtensionPointId() );
}
```

```
    private synchronized void storeServiceIntoClientStateStorage(StatefulService pooled)
    {
                    _stateStorage.store( getServicePoint().getExtensionPointId(), pooled );
    }

    private synchronized StatefulService constructStatefulService()
    {
        try
        {
            Object core = constructCoreServiceImplementation();

            // This is related to bean services.

            if (!_serviceInterface.isInstance(core))
                core = constructBridgeProxy(core);


            return new StatefulService(core);
        }
        catch (Exception ex)
        {
            throw new ApplicationRuntimeException( ex);
        }
    }

    protected void unbindPooledServiceFromCurrentThread(StatefulService pooled)
    {
        _notifier.removeThreadCleanupListener(pooled);

        _activeService.set(null);

        pooled.passivate();

        storeServiceIntoClientStateStorage(pooled);
    }

    /**
     * Invokes {@link #getServiceImplementationForCurrentThread()}to instantiate an instance of the
     * service.
     */
    public void instantiateService()
    {
        getServiceImplementationForCurrentThread();
    }
}
```