DSM for Java

Like | Updated May 02 by David A. Wood | Tags: *None*

Below find a brief overview of some of the key data structures and their relationships. Javadoc is also available.

Introduction

The Distributed State Machine architecture assumes a simple model of a distributed set of application nodes, each maintaining its own local data, and collaborating to solve a common problem by sharing data as necessary. Instead of direct node addressing, network topology relationships such as parent, neighbor, reachable, etc are used to address nodes. These features enable, for example, the implementation of routing algorithms in a mobile ad hoc network through the sharing of neighbors and routes with node neighbors. With this framework in mind, the primary goals of the architecture are to 1) provide topology-based addressing, 2) support dynamic networks, and 3) allow the efficient and intuitive sharing of data.

A distributed hash table-based registry of collaborating application nodes (i.e. peers) is used to maintain the list of *reachable* nodes. From this each node develops its own local set of *neighbors*. To support dynamic networks, the registrations expire and application nodes periodically re-register. This allows the network to partition and the registry to adapt to the changing set of connected nodes. Logical topology relationships such as *parent* and *child* are defined by the application.

Shared data is assigned a unique name defined by the application. The data is then made available to peers via queries that reference one or more named data. Queries are either synchronous or continuous and are addressed to a set of nodes with one or more topological relationships (i.e. neighbor, parent, etc.). For example, a node might request the resources for each virtual machine running on its children (assuming the nodes are sharing such information). Continuous queries may specify a condition that must be met before results are returned. Conditions may be defined on a pair-wise (peer-to-peer) basis or as an aggregation of values across multiple nodes in the topological relationship. For example,

- pair-wise: return the virtual machine resources for those nodes that are exceeding 50% cpu utilization.
- aggregation : return results when the local node's cpu utilization is less than the average of all neighbors.

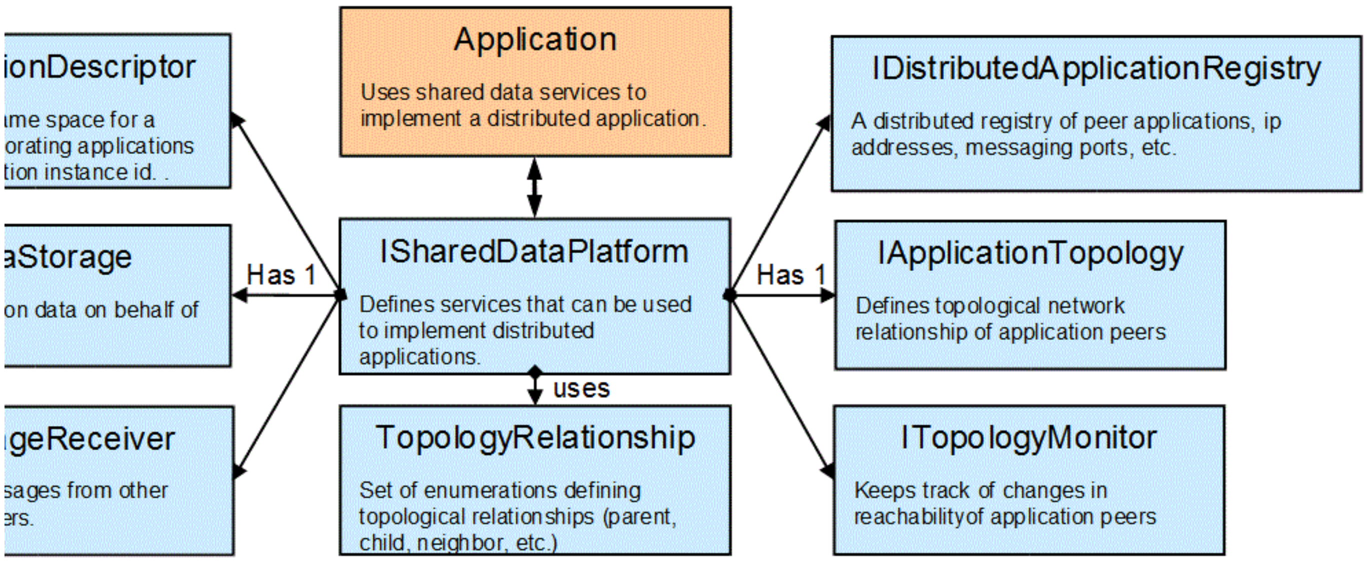
The architecture has been implemented using two distinct shared data types: one for sharing serializable Java objects and the other for sharing rows (tuples) in database tables.

Functional Interfaces used by Shared Data Platform

The key interfaces a developer will make use of are as follows:

- ISharedDataPlatform
- IApplicationDescriptor
- IApplicationTopology
- TopologyRelationship

Additional interfaces (IDataStorage, IMessageReceiver, IDistributedApplicationRegistry, ITopologyMonitor) are discussed here, but generally not used by the developer other than in the construction of the ISharedDataPlatform.


ionDescriptor
ame space for a
orating applications
tion instance id. .
aStorage
on data on behalf of
geReceiver
sages from other
ers.
Has 1
Application
Uses shared data services to implement a distributed application.
ISharedDataPlatform
Defines services that can be used to implement distributed applications.
uses
TopologyRelationship
Set of enumerations defining topological relationships (parent, child, neighbor, etc.)
Has 1
IDistributedApplicationRegistry
A distributed registry of peer applications, ip addresses, messaging ports, etc.
IApplicationTopology
Defines topological network relationship of application peers
ITopologyMonitor
Keeps track of changes in reachabilityof application peers

The ISharedDataPlatform is the central entity that coordinates the sharing of data and may send and/or receive data to/from other shared data applications according to the application's use. It must be started using its *start*() method before any of it services can be used. It is capabile of sending data to other application peers (via IMessageSender, not shown) and receiving data from other applications via IMessageReceiver. To send the data or messages, it uses the IApplicationDescriptor as a destination address. ISharedDataPlatform provides two primary data access methods: synchronous queries and continuous queries whose results are returned asynchronously. The synchronous queries are issued through the *queryRemoteData*() methods and *addRemoteDataListener*() methods are used to create continuous queries. Specific queries and conditions may be set on the queries through the use of IDataQuery and ICondition objects, discussed below.   Continuous queries return their results to an IDataAction interface method, also discussed below. Both types of queries are targeted to application peers via their topological relationship to the querying application, using a TopologyRelationship. For example, queries can be applied to neighbors, parents, children or all reachable peers. Supported relationships are defined by the IApplicationTopology. The ISharedDataPlatform is a generically-typed interface that is intended to be specialized to the type of data that is being shared (ISharedData discussed below).

The IApplicationDescriptor defines an application namespace for the set of applications that are cooperating to manage the distributed data. It also defines an application instance identifier which is unique among the set of cooperating applications. Both the namespace and the identifier are application defined and default values for instance ids can be used. Any application that uses the same application name space will have access to data from peers using the same application name space.

The IDistributedApplicationRegistry is an entity that is used by all cooperating applications. It holds entries for active applications, their instance identifiers and other information required to share messages and data.

IDataStorage provides set of methods needed by the ISharedDataPlatform to store and query shared data (ISharedData).

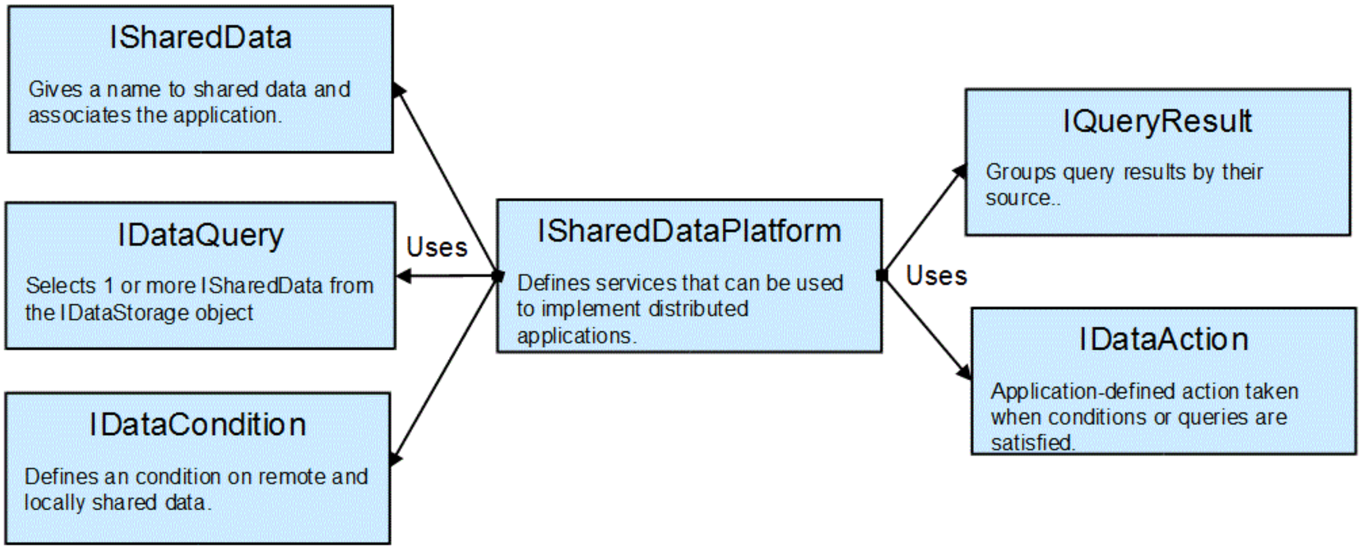
IApplicationTopology defines the network topology relationship of one peer application to another. For example, the topology could define all the parent-child relationships in a tree or the neighbors in an unstructured ad hoc network. This is used when creating queries to identify the nodes to which the queries apply.

ITopologyMonitor is generally not used by application developers, but provides notifications to the ISharedDataPlatform when the population or topology of peer application nodes changes. This allows the ISharedDataPlatform to maintain subscriptions as peer applications appear and disappear a dynamic network environment.

Data Model for ISharedDataPlatform

The key interfaces are as follows:

- ISharedData
- IDataQuery
- IDataCondition
- IDataAction
- IQueryResult


ISharedData
Gives a name to shared data and associates the application.
IDataQuery
Selects 1 or more ISharedData from the IDataStorage object
IDataCondition
Defines an condition on remote and locally shared data.
Uses
ISharedDataPlatform
Defines services that can be used to implement distributed applications.
Uses
IQueryResult
Groups query results by their source..
IDataAction
Application-defined action taken when conditions or queries are satisfied.

The ISharedData is the fundamental data container in the system and assigns a name to the shared data. The name must be unique within the application. It is the ISharedData that is "shared into" the platform via the platforms *setSharedData*() method and that which is returned as the result of a query.

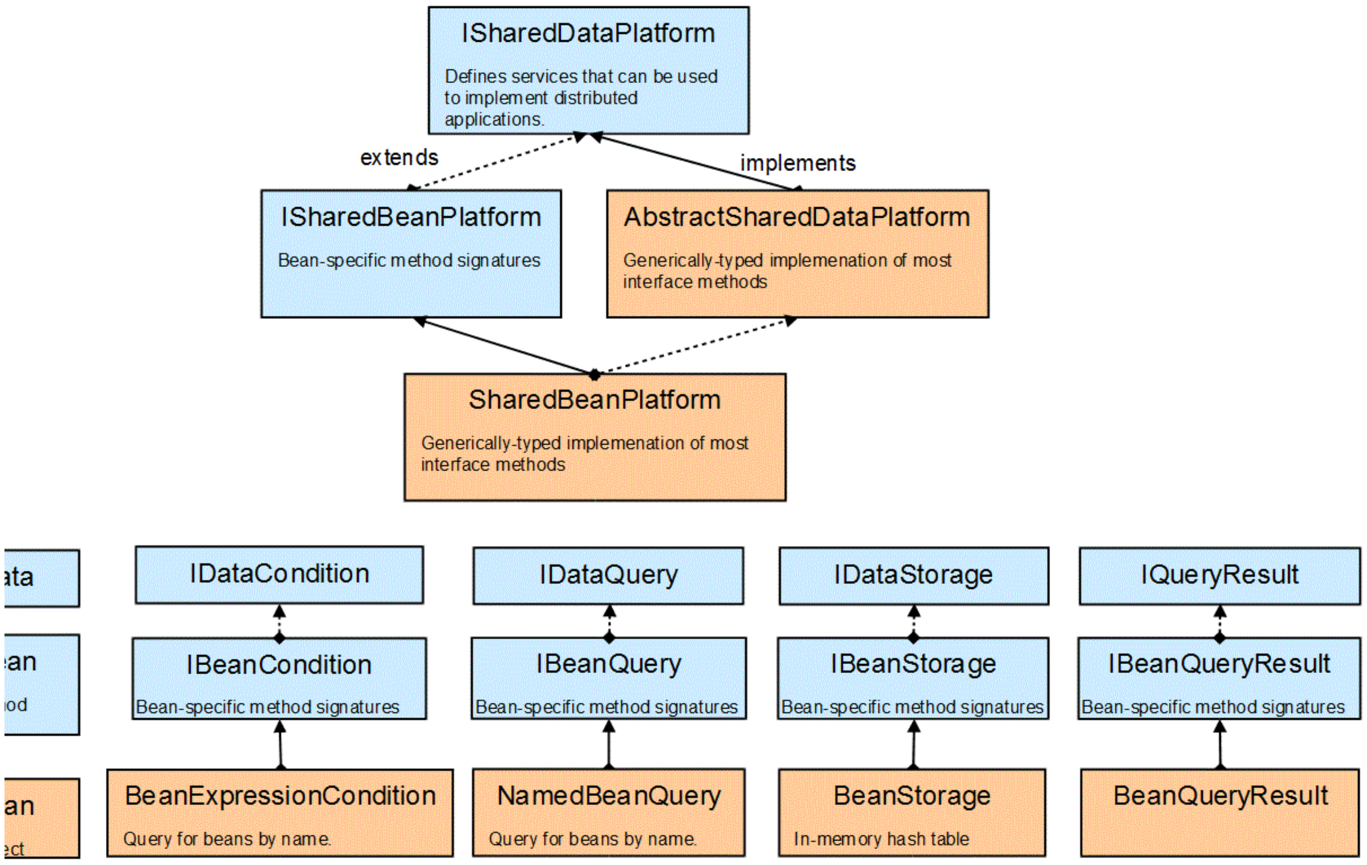
The IDataQuery may be used to select a specific portion of the shared data space. It implements the query and is provided local IDataStore against which to make the query. It can be used, for example, to select rows from a specific database table.

The IDataCondition is used primarily with continuous queries to filter the results returned. Conditions are defined over one or more sets of remote data and optionally the local data. The IDataCondition defines the IDataQuery that must be made to evaluate the condition.

Remote query results are always returned in an IQueryResult object which groups the results by the source of the data. In the case of synchronous queries, this is returned by the *queryRemoteData*() methods. For continuous queries, the IQueryResult is provided to the IDataAction method that was registered with the continuous query. The IDataAction method is called when results, meeting the optional condition, are available.

Each of these interfaces is generically typed and are intended to be extended in a data type-specific manner. The sections below discuss the Java and SQL extensions to and implementation of these interfaces.

Shared Bean Platform and Data


ISharedDataPlatform
Defines services that can be used to implement distributed applications.
extends
implements
ISharedBeanPlatform
Bean-specific method signatures
AbstractSharedDataPlatform
Generically-typed implemenation of most interface methods
SharedBeanPlatform
Generically-typed implemenation of most interface methods
IDataCondition
IBeanCondition
Bean-specific method signatures
BeanExpressionCondition
Query for beans by name.
IDataQuery
IBeanQuery
Bean-specific method signatures
NamedBeanQuery
Query for beans by name.
IDataStorage
IBeanStorage
Bean-specific method signatures
BeanStorage
In-memory hash table
IQueryResult
IBeanQueryResult
Bean-specific method signatures
BeanQueryResult

The key interfaces and classes are as follows:

- ISharedBeanPlatform and SharedBeanPlatform
- ISharedBean and SharedBean
- IBeanQuery and NamedBeanQuery
- IBeanCondition and BeanExpressionCondition
- IBeanQueryResult and BeanQueryResult

For the most part, these interfaces and classes serve to define concrete types for the super classes. In this case, the term *bean* only implies serializable here and as such these classes operate on Java Serializable objects. The SharedBeanPlatform adds a convenience method to change shared bean values as follows:

boolean setBeanData(String name, Serializable obj) throws DSMException;

NamedBeanQuery allows the query of beans by name, as might be expected. BeanStorage is an in-memory, non-persistent, repository for the beans stored with the platform.

Of particular interest is the BeanExpressionCondition which allows scripted definitions of logical expressions on named beans and their data or values. It uses JEXL which provides support for a Java-like expression language. The symbols made available to the expression evaluation context are the following:

Local - a hashtable of bean names mapped to local bean instances. The bean instances are those stored in the ISharedBean.

Peer- a hashtable of bean names mapped to bean instances from a single remote application peer. The bean instances are those stored in the ISharedBean.

Average - provides average values of Java Number values within beans from 1 or more remote peers.

Sum- provides the sum of values of Java Number values within beans from 1 or more remote peers.

Min- provides minimum of values of Java Number values within beans from 1 or more remote peers.

Max- provides maximum of values of Java Number values within beans from 1 or more remote peers.

Count - provides the count of named remote beans.

The *Local* and *Peer* instances are (read-only) Java HashMap instances and the *get*() method can be used to acquire the bean instances and define expressions using them. The aggregator instances (Average, Sum, Min, Max and Count) provide the methods:

Number get(String beanName, String memberName);

Number get(String beanName);

where *memberName* may be a the name of a field within a Java object or a zero-args method name and parens. In either case, the field or method must evaluate to a Number type object or a primitive int, float or double value.

NOTE: the Peer instance and aggregator instances can not be used in the same expression.

As an example, consider the following classes which are shared with the bean name *resources* and *config*:

resources	config

`public class Resources {`
`    double cpuUtilization;`
`    MemoryMap memMap;`
`    int cpuCount();`
`}`
	public class Config {
	double cpuThreshold;
	int megabytesPerVM;
	}

The following expressions can then be defined:

1. Local.get("resources") .cpuUtilization < Peer.get("resources").cpuUtilization
2. Local.get("resources") .memMap.available() < Peer.get("resources").memMap.available()
3. Local.get("resources").cpuCount() < Average.get("resources","cpuCount()")
4. Local.get("config").cpuThreshold < Average.get("resources", "cpuUtilization")
5. Count.get("resources") > 4

Example: Shared Bean Hello World

Here we consider a very simple use of the SharedBeanPlatform to send text strings from one peer to the other. No conditions or queries are used or needed for this simple example, which can be found in the source in DSM-Core/samples/com/ibm/watson/dsm/samples/BeanHelloWorld.java.

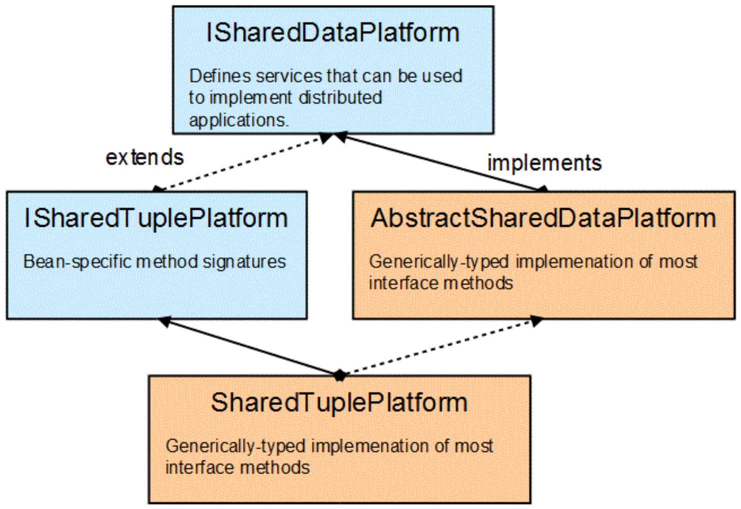
```
public class BeanHelloWorld implements  IBeanAction {
    // One instance per application to provide the data sharing services.
    ISharedBeanPlatform sharedBeanPlatform;
    // IDataStorage implementations must be able to hold data from multiple
distinct applications.
    // Therefore, we can use a single static instance of BeanStorage for all shared
platform instances.
    static IBeanStorage beanStorage = new BeanStorage();
    /**
     * Create an instance containing a shared bean platform that has the given
instance name.
     * The platform is started and  a subscription to all data is created.
     * @param instanceID
     * @throws DSMException
     */
    public BeanHelloWorld(String instanceID) throws DSMException {
        System.out.println("Creating instance " + instanceID);
        // Create the application descriptor for this instance of the platform.
Note that both arguments to
        // this constructor are optional, although in general, you'll want to set
the first to define the
        // application name that peers all share.  Sharing the same name puts them
in the same application
        // for data sharing space.
        IApplicationDescriptor appDesc = new
ApplicationDescriptor("BeanHelloWorld", instanceID);
        // Create the shared bean platform.  All platform instances will use the
same IBeanStorage instance.
        sharedBeanPlatform = new SharedBeanPlatform(appDesc, beanStorage);

        // The platform must be started before anything can really be done with it.
        sharedBeanPlatform.start();

        // Create a subscription to ALL data on reachable nodes in this
application.
        // The subscription is to ALL data because neither an IBeanCondition or
IBeanQuery is specified
        // in the subscription creation request.
        sharedBeanPlatform.addRemoteDataListener(TopologyRelationship.Reachable,
this);
```

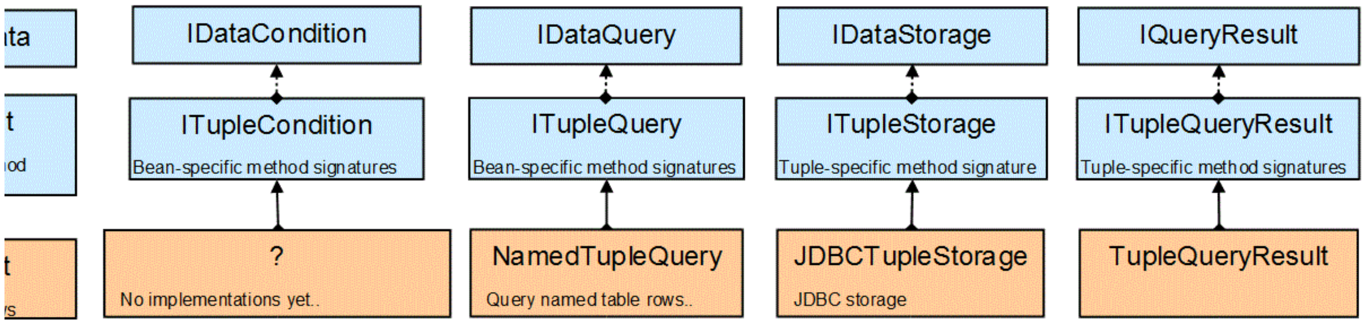
```
    }
    public static void main(String[] args) throws DSMException {
        // Create/initialize 3 platforms all with different instance ids.
        BeanHelloWorld localPlatform = new BeanHelloWorld("local");
        BeanHelloWorld remotePlatform1 = new BeanHelloWorld("remote1");
        BeanHelloWorld remotePlatform2 = new BeanHelloWorld("remote2");
        // Give the registry some time to share its information between platforms
        System.out.print("Waiting for platforms to register...");
        try {Thread.sleep(5000);} catch (InterruptedException e) {    }
        System.out.println("done.");
        // Store a message with the given name.
        localPlatform.sharedBeanPlatform.setBeanData("msg", "Hello World!");
        // Resaving it overwrites the previous value.
        localPlatform.sharedBeanPlatform.setBeanData("msg", "Goodbye World!");
        // Shutdown the platforms in an orderly manner.
        remotePlatform2.sharedBeanPlatform.stop();
        remotePlatform1.sharedBeanPlatform.stop();
        localPlatform.sharedBeanPlatform.stop();
    }
    /**
     * This is the method that is called when we receive subscribed data.
     */
    public void action(List<ISharedBean> localData, List<IBeanQueryResult>
remoteData, IBeanCondition condition) {
        String appInstance =
sharedBeanPlatform.getApplicationDescriptor().getInstanceID();
        StringBuilder sb = new StringBuilder();
        sb.append("Application instance '" + appInstance + "' received action
request with the following data...");
        sb.append("\nLocal data list of size " + localData.size());
        if (localData.size() > 0) {
            for (int i=0 ; i<localData.size() ; i++) {
                ISharedBean sharedBean = localData.get(i);
                sb.append("\nlocalData[" + i + "]=" + sharedBean.getBean());
            }
        }
        sb.append("\nReceived results from " + remoteData.size() + " peer(s)");
        if (remoteData.size() > 0) {
            for (int i=0 ; i<remoteData.size() ; i++)
                sb.append("\nPeer '" + remoteData.get(i).getSource() +
                            "' sent the following list of results  " +
remoteData.get(i).getResults());
        }
        sb.append("\n");
        System.out.println(sb.toString());
    }
}
```

If you run this, you will see the following

```
Watson Policy Management Library - Copyright IBM Corporation 2008,2011.
Copyrights also by Apache Software Foundation (http://apache.org) and Antlr (http://antlr2.org)
Appending properties file to logging configuration: file:/C:/dev/concert-ws/generic/DSM-Core/dsm.properties
Creating instance local
Creating instance remote1
Creating instance remote2
Waiting for platforms to register...done.
Application instance 'remote1' received action request with the following data...
Local data list of size 0
Received results from 1 peer(s)
Peer 'name=BeanHelloWorld id=local' sent the following list of results  [Hello World!]
Application instance 'remote2' received action request with the following data...
Local data list of size 0
Received results from 1 peer(s)
Peer 'name=BeanHelloWorld id=local' sent the following list of results  [Hello World!]
Application instance 'remote1' received action request with the following data...
Local data list of size 0
Received results from 1 peer(s)
Peer 'name=BeanHelloWorld id=local' sent the following list of results  [Goodbye World!]
Application instance 'remote2' received action request with the following data...
Local data list of size 0
Received results from 1 peer(s)
Peer 'name=BeanHelloWorld id=local' sent the following list of results  [Goodbye World!]
```

Shared Tuple Platform and Data


ISharedDataPlatform
Defines services that can be used to implement distributed applications.
extends
implements
ISharedTuplePlatform
Bean-specific method signatures
AbstractSharedDataPlatform
Generically-typed implemenation of most interface methods
SharedTuplePlatform
Generically-typed implemenation of most interface methods



The key implementations are as follows:

- ISharedTuplePlatform and SharedTuplePlatform
- ITupleSet and TupleSet
- ITupleQuery and NamedTupleQuery
- ITupleQueryResult and TupleQueryResult

As for the shared bean platfor,m, for the most part, these interfaces and classes serve to define concrete types for the super classes, except that a data framework for tuples (or rows of a database table) is also included here (but not shown yet). The TupleSet provides the contents of selected rows from a single database table. The NamedTupleQuery provides the ability to query all rows from a named table. Certainly addition queries can be develop to, for example, select specific rows of a table or join two or more tables. The JDBCTupleStorage provides the underlying implementation needed to store ITupleSets within the platform. It can be configured using the dsm.properties file to use any JDBC URL or driver. At this time, we have primarily used the in-memory implementation provided by Derby, but we have also used SolidDB and persistent Derby tables.

Example: Shared Tuple Hello World

The following is a very simple use of the shared platform to send text messages. This is not intended to motivate the use of the framework, but to just show the mechanics of sharing tuples. The following example can be found in the DSM-Core project under samples/com/ibm/watson/dsm/samples/TupleHelloWorld.java.

```
public class TupleHelloWorld implements  ITupleAction {

    // One instance per application to provide the data sharing services.
    ISharedTuplePlatform sharedTuplePlatform;

    // IDataStorage implementations must be able to hold data from multiple
distinct applications.
    // Therefore, we can use a single static instance of ITupleStorage for all
shared platform instances.
    // Although the platform only needs an ITupleStorage instance, we use the
JDBCTupleStorage instance
    // to create the needed table (which ITupleStorage can not do).
    static JDBCTupleStorage tupleStorage = new
JDBCTupleStorage(DSMProperties.instance().getProperties());

    final static String MSG_TABLE_NAME = "msg";

    /**
     * Create an instance containing a shared tuple platform that has the given
instance name.
     * The platform is started and  a subscription to all data is created.
     * @param instanceID
     * @throws DSMException
     */
    public TupleHelloWorld(String instanceID) throws DSMException {
        System.out.println("Creating instance " + instanceID);

        // Create the application descriptor for this instance of the platform.
Note that both arguments to
        // this constructor are optional, although in general, you'll want to set
the first to define the
        // application name that peers all share.  Sharing the same name puts them
in the same application
        // for data sharing space.
        IApplicationDescriptor appDesc = new
ApplicationDescriptor("TupleHelloWorld", instanceID);

        // Create the shared tuple platform.  All platform instances will use the
same ITupleStorage instance.
        sharedTuplePlatform = new SharedTuplePlatform(appDesc, tupleStorage);

        // The platform must be started before anything can really be done with it.
        sharedTuplePlatform.start();

        // Create the table to store our messages
        tupleStorage.createStorage(appDesc, MSG_TABLE_NAME, "CREATE TABLE " +
MSG_TABLE_NAME + " (text varchar(128))");
```

```
        // Create a subscription to ALL data on reachable nodes in this
application.
        // The subscription is to ALL data because neither an ITupleCondition or
ITupleQuery is specified
        // in the subscription creation request.
        sharedTuplePlatform.addRemoteDataListener(TopologyRelationship.Reachable,
this);
    }
    public static void main(String[] args) throws DSMException {
        // Create/initialize 3 platforms all with different instance ids.
        TupleHelloWorld localPlatform = new TupleHelloWorld("local");
        TupleHelloWorld remotePlatform1 = new TupleHelloWorld("remote1");
        TupleHelloWorld remotePlatform2 = new TupleHelloWorld("remote2");
        // Give the registry some time to share its information between hw1 and hw2
        System.out.print("Waiting for platforms to register...");
        try {Thread.sleep(5000);} catch (InterruptedException e) {    }
        System.out.println("done.");

        // Store a message.
        ITupleSetDescriptor desc =
tupleStorage.getDescriptor(localPlatform.sharedTuplePlatform.getApplicationDescriptor(),

MSG_TABLE_NAME);
        ITupleSet tset = new TupleSet(desc);
        tset.add(new Tuple("Hello World!"));
        localPlatform.sharedTuplePlatform.setSharedData(tset);
        // Resaving it overwrites the previous value.
        tset.clear();
        tset.add(new Tuple("Goodbye World!"));
        localPlatform.sharedTuplePlatform.setSharedData(tset);
        // Now append a value to the MSG table and get notified with both Goodbye
messages.
        tset.clear();
        tset.add(new Tuple("Goodbye World! (again)"));
        localPlatform.sharedTuplePlatform.appendTuples(tset);

        // Shutdown the platforms in an orderly manner.
        remotePlatform2.sharedTuplePlatform.stop();
        remotePlatform1.sharedTuplePlatform.stop();
        localPlatform.sharedTuplePlatform.stop();
    }

    /**
     * This is the method that is called when we receive subscribed data.
     */
    public void action(List<ITupleSet> localData, List<ITupleQueryResult>
remoteData, ITupleCondition condition) {
        String appInstance =
sharedTuplePlatform.getApplicationDescriptor().getInstanceID();
        StringBuilder sb = new StringBuilder();
        sb.append("Application instance '" + appInstance + "' received action
request with the following data...");
        sb.append("\nLocal data list of size " + localData.size());
        if (localData.size() > 0) {
            for (int i=0 ; i<localData.size() ; i++) {
                ITupleSet sharedTuple = localData.get(i);
                sb.append("\nlocalData[" + i + "]=" + sharedTuple);
            }
        }
        sb.append("\nReceived results from " + remoteData.size() + " peer(s)");
        if (remoteData.size() > 0) {
            for (int i=0 ; i<remoteData.size() ; i++)
                sb.append("\nPeer '" + remoteData.get(i).getSource() +
                            "' sent the following list of results  " +
remoteData.get(i).getResults());
        }
        sb.append("\n");
        System.out.println(sb.toString());
    }
}
```

When this is run, the following is printed out (bold added here)...

Loading dsm properties from file:/C:/dev/concert-ws/generic/DSM-Core/dsm.properties
Creating instance **local**
Watson Policy Management Library - Copyright IBM Corporation 2008,2011.
Copyrights also by Apache Software Foundation (http://apache.org) and Antlr (http://antlr2.org)
Loading wpml properties from file:/C:/dev/concert-ws/generic/DSM-Core/wpml.properties
Appending properties file to logging configuration: file:/C:/dev/concert-ws/generic/DSM-Core/wpml.properties
...
Creating instance **remote1**
...
Creating instance **remote2**
...
Waiting for platforms to register...done.
Application instance '**remote2**' **received** action request with the following data...
Local data list of size 0
Received results from 1 peer(s)
Peer 'name=TupleHelloWorld id=local' sent the following list of results [TupleSet:TupleSetDescriptor:
appDesc=name=TupleHelloWorld id=remote2, name=msg, [ColumnDescriptor: name=TEXT, type=String][
Tuple[TupleEntry: value**=Hello World!**]
]]

Application instance **'remote1' received** action request with the following data...
Local data list of size 0
Received results from 1 peer(s)

```
Peer 'name=TupleHelloWorld id=local' sent the following list of results  [TupleSet:TupleSetDescriptor:
appDesc=name=TupleHelloWorld id=remote2, name=msg, [ColumnDescriptor: name=TEXT, type=String][
Tuple[TupleEntry: value=Hello World!]
]]

Application instance 'remote2' received action request with the following data...
Local data list of size 0
Received results from 1 peer(s)
Peer 'name=TupleHelloWorld id=local' sent the following list of results  [TupleSet:TupleSetDescriptor:
appDesc=name=TupleHelloWorld id=remote2, name=msg, [ColumnDescriptor: name=TEXT, type=String][
Tuple[TupleEntry: value=Goodbye World!]
]]

Application instance 'remote1' received action request with the following data...
Local data list of size 0
Received results from 1 peer(s)
Peer 'name=TupleHelloWorld id=local' sent the following list of results  [TupleSet:TupleSetDescriptor:
appDesc=name=TupleHelloWorld id=remote2, name=msg, [ColumnDescriptor: name=TEXT, type=String][
Tuple[TupleEntry: value=Goodbye World!]
]]

Application instance 'remote2' received action request with the following data...
Local data list of size 0
Received results from 1 peer(s)
Peer 'name=TupleHelloWorld id=local' sent the following list of results  [TupleSet:TupleSetDescriptor:
appDesc=name=TupleHelloWorld id=remote2, name=msg, [ColumnDescriptor: name=TEXT, type=String][
Tuple[TupleEntry: value=Goodbye World!]
Tuple[TupleEntry: value=Goodbye World! (again)]
]]

Application instance 'remote1' received action request with the following data...
Local data list of size 0
Received results from 1 peer(s)
Peer 'name=TupleHelloWorld id=local' sent the following list of results  [TupleSet:TupleSetDescriptor:
appDesc=name=TupleHelloWorld id=remote2, name=msg, [ColumnDescriptor: name=TEXT, type=String][
Tuple[TupleEntry: value=Goodbye World!]
Tuple[TupleEntry: value=Goodbye World! (again)]
]]
```

DSM Rules Engine

The DSM Rules engine is an application built on top of the ISharedTuplePlatform. It uses rules to manipulate the local values of its tables. The rules can trigger the exchange of tuples with other rules engines, which is facilitated by the ISharedTuplePlatform.

Comments

There are no comments.