Distributed Computing SDK – Programmer's Guide

Version 2.1.0
March 2012

1 SDK Overview

1.1 Synopsis

This document gives you information about the theory behind the software, the growing need for adaptive networks as well as a complete overview of the Adaptinet Distributed Computing Software Development Kit and how you can architect distributed applications using it. We describe the internal components of the software including the TransCeiver, its API and the development and configuration tools included in the SDK.

1.2 Adaptive Networks

Within the realm of distributed environments the need for networks that adapt autonomously to dynamic conditions has become apparent. At Adaptinet we have created self-adaptive networks that can be optimized by means of agents residing at each node of the network. The capability of these agents is a set of active rules. The algorithms apply these rules in the face of dynamically evolving conditions.

As distributed networks become larger and more complex, the need for managing them effectively, optimizing their capacity and reducing their operation costs will become a priority. This will become more urgent as larger applications of distributed networks begin to roll out.

Adaptinet's Distributed Computing SDK uses Software Agents at the Node Level

There are several common ways to build distributed applications. What is lacking is the ability to adjust to growth and adapt to traffic patterns.

In contrast Adaptinet uses software agents that reside on each node and optimize the network in real-time. These agents require no human intervention hence the term adaptive. While load conditions change and nodes connect and disconnect, the network can still run at a near optimal state. The network accomplishes this without any operator intervention. At the node level data is continually collected and the network is probed for current conditions. This information is used through a series of algorithms to continually adjust the topology of the nodes within the network.

There is no need to keep global knowledge of the topology in the network; all information is of a local nature. Each node maintains knowledge of a self-defined size of its neighborhood.

Our model is a fairly simple yet very powerful one. The main goal of our network model is the ability to broadcast a message to every node on the network, once and only once. This being accomplished the network can distribute information efficiently, accelerate processing and increase throughput in ways never expected. A search for a resource through a network of many millions of nodes only requires a minimum number of hops. Depending on network conditions this would allow for sub second response times.

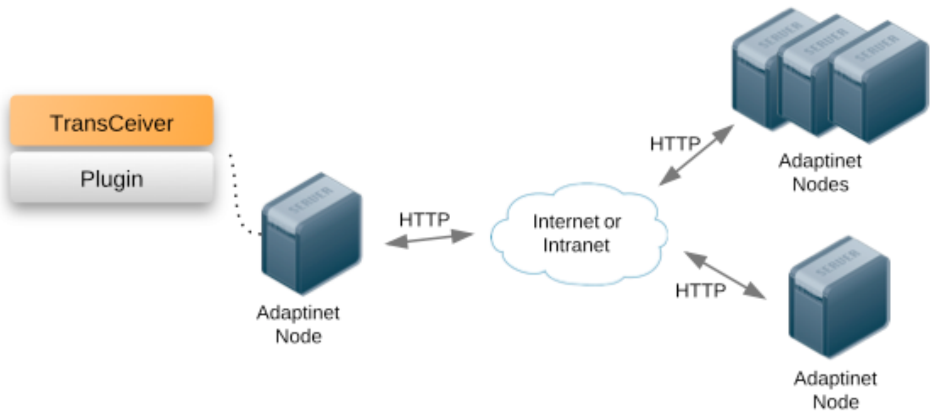
1.3 The Distributed Computing SDK

The Adaptinet Distributed Computing SDK is a platform for building distributed applications. At the heart of the SDK is the TransCeiver, which is both a receiver and transmitter of information. The TransCeiver handles all of the basic communications tasks including listening, parsing, message routing and message transmission.

Many distributed applications can be easily built and then docked (or attached) to the Adaptinet TransCeiver. Adaptinet applications are called plug-ins and once docked (or plugged in), these applications can receive, process and send information onto the next node or back to the calling node. The Adaptinet Distributed Computing SDK provides developer tools for easily creating plug-ins for the TransCeiver.

The Adaptinet TransCeiver is a small (~200K) Java program that works with plug-ins, which are written in Java. The TransCeiver listens for HTTP messages. It can also post HTTP requests to other TransCeivers. The TransCeiver's main tasks are to host distributed applications and facilitate communication between these applications (held within as plug-ins) and other TransCeivers.

The protocol used for communication is HTTP and XML. The use of HTTP alleviates firewall issues within corporate environments while the use of XML



increases the extensibility of the TransCeiver to communicate with other types of transceivers and / or servers. Adaptinet uses a very simple XML protocol called AdaptX, which is also available with the Distributed Computing SDK. Note: The Adaptinet TransCeiver also supports HTTPS and SSL.

Adaptinet plug-ins communicate with one-another by sending HTTP messages to each other. This functionality is nicely hidden within the TransCeiver API. The receiving TransCeiver takes the body of the HTTP POST, which is an AdaptX message expressed in XML, and delivers it to the plug-in targeted within the XML message (see the AdaptX protocol for more detail).

The TransCeiver is a fully threaded application, allowing a number of simultaneous connections with remote nodes. The Adaptinet Distributed Computing SDK infrastructure provides a platform that will easily extend existing applications and give rise to a new class of distributed applications. The SDK is a robust base platform that provides the largest part of the infrastructure needed for developing distributed applications.

1.4 Features of the Distributed Computing SDK

The Adaptinet Distributed Computing SDK is a robust platform for quickly building distributed applications. Some of the features include:

- **Adaptive** – Once rolled out, the Adaptinet network is automatically and continually load balanced.
- **Efficient Broadcasting** – The advanced architecture ensures that messages are broadcast to every node on the network once and only once.
- **Lightweight** – The Adaptinet TransCeiver is small, facilitating its use on a variety of systems.
- **Modular** – The TransCeiver kernel can easily be extended with modular plug-ins.
- **Open Communication Standards** – The Distributed Computing SDK uses HTTP as the base protocol.
- **Java-based Kernel** – The kernel is 100% Java allowing effortless deployment across multiple platforms.
- **Multithreaded** – Multithreading permits a number of simultaneous connections with foreign clients.
- **Secure** - Support of the Secure Socket Layer (SSL) protocol ensures an encrypted conversation.
- **XML-Based Protocol** – The Distributed Computing SDK makes use of the AdaptX protocol, which is an XML-based protocol.
- **Easily Configurable** – Included within the Distributed Computing SDK is a HTML-based configuration utility supporting real-time changes to the TransCeiver.

2 Installing the Distributed Computing SDK

2.1 Downloading & Setup

The Adaptinet Distributed Computing SDK is available on our web site at http://adaptinet.com/download. Once you have downloaded the software, run the setup program and follow the installation wizard.

There are a number of files and folders/directories created during the installation of the Distributed Computing SDK:

File or Directory	Description
admin/	Contains administration files
samples/	Contains plug-in samples
www/	HTTP server default root
javadocs/	Contains all javadocs
readme.txt	Latest information relating to the install
license.txt	The GNU General Public License for the Adaptinet Distributed Computing SDK
adaptinet.jar	JAR containing binaries for the Adaptinet Distributed Computing SDK
source.jar	Source code for the Adaptinet Distributed Computing SDK
samples.jar	JAR file containing samples
defaultpeers.xml	The default peer file for the TransCeiver
peers.xml	The current peer file for the TransCeiver
transceiver.bat	Windows batch file for starting the TransCeiver
transceiver.sh	Script file for starting the TransCeiver
transceiver.properties	TransCeiver configuration file

2.2 Make Sure the Distributed Computing SDK was Installed Properly

To make sure the SDK was properly installed and functioning we will start the TransCeiver. Please take the following steps:

1. Make sure that a Java Runtime Environment (JRE) 5 or greater has been installed. You can download a JRE or JDK from Sun directly at http://www.java.com.
2. Start the TransCeiver from the shell script included with the installation script transceiver.sh (transceiver.bat for Windows) in the bin/ directory.
3. A TransCeiver console should load displaying information about the TransCeiver (IP, port, version, etc).
4. Your Adaptinet TransCeiver is now working properly. To test a ping to yourself, continue to step 5.
5. The Adaptinet RunTime Console (a GUI application) should also load. If this GUI fails to load, check the transceiver.properties file included with the installation. The showconsole property should be set to true (e.g. showconsole=true).
6. In the RunTime Console, change the Peer List item to point to port 8082

(our machine). The Peer List item should be changed to 127.0.0.1:8082 (the local TransCeiver is running on port 8082).

7. Press the Ping button. You should see a ping from yourself to yourself.
8. Congratulations, you have just sent a message to yourself and received a response from yourself.
9. You can setup the Distributed Computing SDK on other machines (or other ports on the same machine) on the network and attempt to ping those TransCeivers as well.

3 Plugins

3.1 Overview

Adaptinet distributed applications are called “plug-ins” and once docked (or plugged in) to the TransCeiver, these applications can receive, process and send information onto the next node or back to the calling node.

3.2 Writing Plug-ins

To write a Java plug-in for the Adaptinet TransCeiver, you will need to understand the functionality behind three simple Java classes.

- **Plugin** – The Plugin class is an abstract class that is extended for the creation of the base plug-in.
- **Address** – The Address class contains information about the sender or destination of messages that are transferred between plug-ins.
- **Message** – The Message class contains the actual message data transferred between plug-ins.

3.2.1 The Plugin Class

At the heart of an Adaptinet Plug-in is the Plugin class. Plug-ins are created by simply extending this class. The Plugin class provides all of the facilities for plug-in initialization and destruction, error handling and node-to-node communication. Using the communication methods (PostMessage and BroadcastMessage) messages can be sent directly to specific nodes or broadcast to some or all of the nodes within the Adaptinet Distributed Computing network.

Plugin Class
com.adaptinet.pluginagent.Plugin

```
public abstract class Plugin extends PluginRoot
{
   public abstract void init();
   public abstract void cleanup();
   public void unload();
   public void shutdown();
   public void error(String uri, String errorMsg);
   public void peerUpdate();
   public final void localPostMessage(Message message, Object[] args)
         throws PluginException;
   public final void postMessage(Message message, Object[] args);
         throws PluginException;
   public final void broadcastMessage(Message message, Object args[]);
         throws PluginException;
   public boolean preProcessMessage(Envelope env);
   public final Envelope peekMessage(boolean bRemove);
   public final Iterator getPeers(boolean all);
}
```

See the JavaDocs for more details.

3.2.2 The Message Class

The Message class is used by a plug-in when communicating (posting, sending or broadcasting) to other nodes within the Adaptinet Distributed Computing network. Typically, the Message object is used when you are about to create and send a message or to acquire information related to a recently received message.

In preparation for communicating with a remote-node, a Message object is created and contains an Address object, which contains the remote host name, port, plug- in name and method to be called. It may also optionally contain a reply-to Address object. Alternatively, when a message is received from a remote node, the Message object member of the Plugin (msg) contains information about the sending node.

Note about broadcasting: When broadcasting to nodes, the Message object may also contain a "hop count" which determines the number of hops the message will be broadcast. If you have a large network and would like to limit the broadcast to a small set, the "hop count" can be used to control the number of peers that receive the broadcast. For example: by setting a "hop count" to 2, the message will only be broadcast to my immediate child nodes and their immediate child nodes or 2 hops.

Message Class

com.adaptinet.messaging.Message;

```
public final class Message
{
   // Constructors public Message();
   public Message(Address toAddress);
   public Message(String toUri, ITransceiver replyTransceiver);
   public Message(String uri);
   public Message(Message msg);
   public Message(ITransceiver transceiver);
   public Message(ITransceiver transceiver, boolean bSecure);

   //Public Instance Methods
   public Address getAddress();
   public void setAddress(Address address);
   public void setReplyTo(Address replyTo);
   public Address getReplyTo();
   public String getID();
   public void setID(String id);
   public String getTimeStamp();
   public void setTimeStamp();
   public void setTimeStamp(String timestamp);
   public String getHops();
   public void setHops(String hops);
   public void setHops(int hops);
   public int getHopCount();
   public String getCertificate();
   public void setCertificate(String certificate);
   public String getKey();
   public void setKey(String key);
   public void setKey(int key);
   public int getKeyType();
   public void setMethod(String method);
   public void setPlugin(String plugin);
   static public Message createReply(Message msg);
   public Address hop();
   public Address getRoute();
}
```

See the JavaDocs for more details.

3.2.3 The Address Class

The Address class contains information about a node address. Addresses are typically encapsulated within the Message class and the Plugin class. The Message class uses Address objects to define senders and reply-to node addresses. The Plugin class stores the "reply-to" Address of the most recent message received.

Address Class

com.adaptinet.messaging.Address;

```
public final class Address
{
   // Constructors public
   Address();
   public Address(String uri);
   public Address(Address address);
   public Address(ITransceiver transceiver);
   public Address(ITransceiver transceiver, boolean bSecure);

   // Public Instance Methods
   public String getURL(); public
   String getURI();
   public void setURI(String uri); public
   void    setURL(String    url);    public
   boolean isSecure();
   public void setSecure(boolean bSecure);
   public void setPrefix(String prefix);
   public String getPrefix();
   public String getHost(); public
   String getPort();
   public void setPlugin(String plugin);
   public String getPlugin();
   public void setMethod(String method);
   public String getMethod();
   public    void    setType(String    type);
   public    void    setType(String    type);
   public String getType();
```

```
    public int hashCode();
    public void lock(); public
    void unlock();
    public Address getRoute();
    public void setRoute(Address route);
    public Address hop();
    public boolean lastStop();
}
```

See the JavaDocs for more details.

3.3 HelloWorld – A Simple Plug-in

A plug-in is easily created by extending the Adaptinet Plugin class. Let's walk through the HelloWorld sample included with this installation to get a better feel for writing Adaptinet plug-ins.

The HelloWorld plug-in is a simple program that can send a "hello" to another transceiver or to itself if so desired. The plug-in has a GUI that accepts a destination (IP or host name and port) and when the Say Hello button is pressed, sends a simple text message.

This sample consists of two java classes, one which extends the Plugin class (HelloWorld.java) and another (HelloWorldFrame.java) which is the Frame for the HelloWorld GUI.

HelloWorld.java

```
package helloworld;

import javax.swing.*;
import java.awt.*;
import com.adaptinet.pluginagent.Plugin;
import com.adaptinet.messaging.Message;
import com.adaptinet.messaging.Address;

public class HelloWorld extends Plugin
{
    boolean packFrame = false;
    HelloWorldFrame frame = null;

    // This method is called by the transceiver for initialization
    public void init()
    {
        frame = new HelloWorldFrame();

        if (packFrame)
            frame.pack();
        else
            frame.validate();
        frame.setVisible(true);
        frame.setPlugin(this);

        // Set the title of the frame to include my hostname and port
        Address address = new Address(this.transceiver);
        frame.setTitle("Hello World - " + address.getHost() + ":" +
            address.getPort());
    }

    // This method is called by the transceiver for cleanup
    public void cleanup()
    {
    }

    // This method is called by the Frame when the Say Hello button
    // is pressed.
    //
    public void sayHelloButton(String to)
    {
        // Create the argument list for the call to another transceiver
        Object[] args = new Object[1];

        // Create an Address for my local machine by passing in the
        // transceiver object
        Address address = new Address(this.transceiver);
        args[0] = "Hello, my name is " + address.getHost() + ":" +
            address.getPort() + ".";

        // The Message object defines who we are going to talk to.  The
        // URI format is as follows:
        //      http://ip-or-host-name:port/plugin/method
        Message message = new Message("http://" + to +
               "/HelloWorld/Hello", transceiver);

        try
        {
            // Go ahead and send the message
            postMessage(message, args);
        }
        catch(Exception e)
        {
            JOptionPane.showMessageDialog(null, e.getMessage(),
                "Exception", JOptionPane.ERROR_MESSAGE);
        }
    }

    // This method is called by other transceivers running the
    // HelloWorld plugin.
    public void Hello(String text)
    {
        // Pop up a dialog showing the text sent to me
        JOptionPane.showMessageDialog(null, "[Message] " + text,
            "New Message Received", JOptionPane.INFORMATION_MESSAGE);
    }
}
```

First things first, a plug-in is created by extending the Adaptinet Plugin class. The Plugin class is an abstract class containing methods and properties required by the TransCeiver.

```
import com.adaptinet.pluginagent.Plugin;
import com.adaptinet.messaging.Message;
import com.adaptinet.messaging.Address;

public class HelloWorld extends Plugin
{
       ...
}
```

The init() method is used for initialization of the plug-in and is called when the plug- in is loaded by the Transceiver. This method is only called once during the lifetime of the plug-in. In the HelloWorld example above, we use the init() method to initialize and display the Frame for the plug-in.

After initialization occurs, the user can press the Say Hello button on the HelloWorld GUI and the Frame class calls the sayHelloButton(…) method passing a text field containing the address of the desired transceiver to talk to.

```
public void sayHelloButton(String to)
{
    // Create the argument list for the call to another transceiver
    Object[] args = new Object[1];

    // Create an Address for my local machine by passing in the
    // transceiver object
    Address address = new Address(this.transceiver);
    args[0] = "Hello, my name is " + address.getHost() + ":" +
        address.getPort() + ".";

    // The Message object defines who we are going to talk to.  The
    // URI format is as follows:
    //      http://ip-or-host-name:port/plugin/method
    Message message = new Message("http://" + to +
        "/HelloWorld/Hello", transceiver);

    try
    {
        // Go ahead and send the message
        postMessage(message, args);
    }
    catch(Exception e)
    {
        JOptionPane.showMessageDialog(null, e.getMessage(),
            "Exception", JOptionPane.ERROR_MESSAGE);
    }
}
```

The goal of this method is to send a message to another TransCeiver. The way this is done is by calling a method within the HelloWorld plug-in running on a remote TransCeiver. The method that will be called on the remote TransCeiver in this example is named Hello(…). The postMessage() function is used to call this remote method. Before we get to the postMessage, lets review how to setup for a postMessage call. As you can see, the postMessage method takes two parameters, an argument list and a Message object.

The argument list is nothing more than an array of Objects. In this case, the method that we intend to call on the remote TransCeiver (Hello) takes only one parameter (a String). We pack a simple message into the string identifying the local machine as the caller.

The Message object defines whom we are going to send the message to and can be initialized with a simple URI.

```
Message message = new Message("http://" + to +
      "/HelloWorld/Hello", transceiver);
```

In this case, we are going to use the HTTP protocol to call the remote method Hello on the remote plug-in named HelloWorld. The “to” string defines the remote hostname (or IP) and port number. The port number corresponds to the port upon which the remote TransCeiver is listening.

Now we are ready to call the postMessage method. The postMessage breaks down the call into an XML message and posts it to the remote TransCeiver using the HTTP protocol. If the destination TransCeiver is not listening or is down, an AdaptinetException will be thrown.

When the message is received by the remote TransCeiver, the HelloWorld Hello(…) method is called with the parameter passed from the original TransCeiver. The exchange is complete and the remote TransCeiver pops up a dialog box displaying the message received.

```
public void Hello(String text)
{
    // Pop up a dialog showing the text sent to me
    JOptionPane.showMessageDialog(null, "[Message] " + text,
        "New Message Received", JOptionPane.INFORMATION_MESSAGE);
}
```

Once you get the hang of it, you can see how easy it is to build distributed applications using the Adaptinet Distributed Computing SDK. You can easily write a distributed application that shares information with other nodes, distributes a workload or anything else you can think of. The beauty of the Distributed Computing SDK is the ability to call any method on any plugin within a distributed Adaptinet network.

Please refer to the JavaDocs included with this package for more details regarding the Adaptinet Distributed Computing SDK classes. Also refer to some of the other samples included with this package for help with the extended features of the Adaptinet Distributed Computing SDK.

3.4 Deploying the HelloWorld Sample

To deploy the HelloWorld sample plug-in included with this installation, follow the outlined procedure:

1. Modify the plugins.xml file located in the ./adaptinet directory as shown below:

```
<?xml version="1.0"?>
<Plugins>
   <Plugin>
      <Name>HelloWorld</Name>
      <Type>helloworld.HelloWorld</Type>
      <Classpath>c:\adaptinet\samples</Classpath>
      <Preload>1</Preload>
   </Plugin>
</Plugins>
```

   Make sure the <Classpath> element has the proper classpath for the HelloWorld plug-in. Also make sure the <Preload> element is set to 1.

   NOTE: You could alternatively use the *Administration Console* to register the plug-in. See the *Administrative Console* section in this guide for more details.

2. If you are running the TransCeiver from a batch file or shell file, modify the class path to point to the sample program as well as the Adaptinet Transceiver.

```
java -cp .\samples\helloworld;.\adaptinet.jar
   com.adaptinet.transceiver.Transceiver -config transceiver.properties
```

3. Run it.

3.5 Advanced Plug-in Message Processing

To fully understand how messages are routed through a TransCeiver to a plug-in we will take a detailed tour of their journey. When a plug-in is started, a queue is created on its behalf. This plug-in queue is used to store messages that have been posted to the plug-in. When a message is first received it is handed to the appropriate mime handler. Here the messages are parsed and sent to the appropriate plug-in agent. It is the responsibility of the agent to maintain and synchronize the message queue for a plug-in. As messages are received the agent will push them onto a queue for subsequent processing. In turn, each message is popped off the queue and dispatched to the appropriate method of the handler.
This callback-based metaphor will be appropriate for most types of applications. If special processing is required, a plug-in author can override two methods of the Plugin base class.

preProcessMessage(Envelope env)

The plug-in message loop will call this method prior to routing the messages to the plug-in. The default method simply returns true allowing the message processing to deliver the message to the appropriate method of the plug-in. If the plug-in requires special processing it can override preProcessMessage and do any processing prior to the message being dispatched. If the plug-in wished to stop the processing of this message, it simply should return false and delivery will not be attempted. If a plug-in has no need to handle messages prior to their delivery it should not attempt to override this method.

peekMessage(boolean bRemove)

A plug-in author can use this method to preview the next message in the queue. This does not remove the message from the queue and it will be processed by the message loop as normal. If the loop has not processed this message a subsequent call will return the same message. Depending on the processing requirements the message delivery can be interrupted. By passing in true for bRemove parameter
the message will be deleted from the queue. Normally, a message will be retrieved with peekMessage passing in bRemove as false. The plug-in would examined the message and determine whether normal processing should continue. If the message needs to be removed from the queue a subsequent call to peekMessage is necessary. Calling peekMessage with bRemove as true will remove it form the queue and it will never be dispatched to the plug-in.

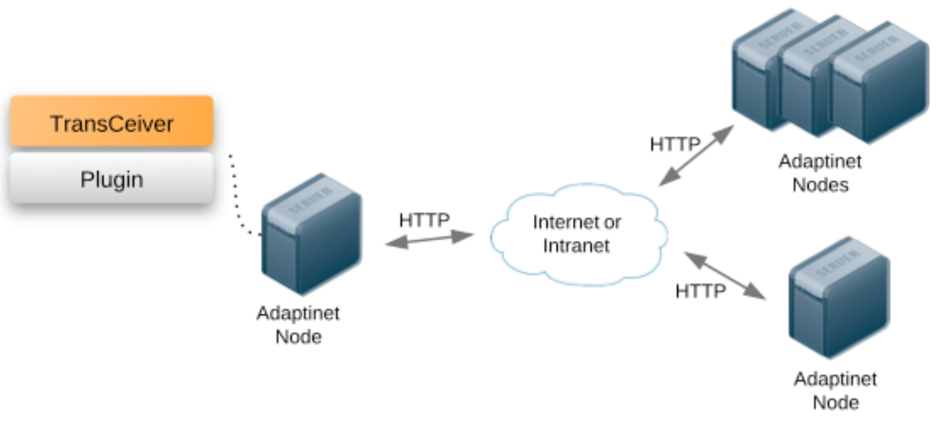
4 Architecting Distributed Applications

There are a number of ways to architect an Adaptinet distributed application with the Distributed Computing SDK. You can distribute your application as a plug-in or series of plug-ins bundled with the TransCeiver or possibly with multiple TransCeivers in special cases. Typically, a TransCeiver is dedicated to your application, which will be listening on a specific port on the machine. The TransCeiver is relatively small (~200K) enabling each Adaptinet application to use an independent TransCeiver.

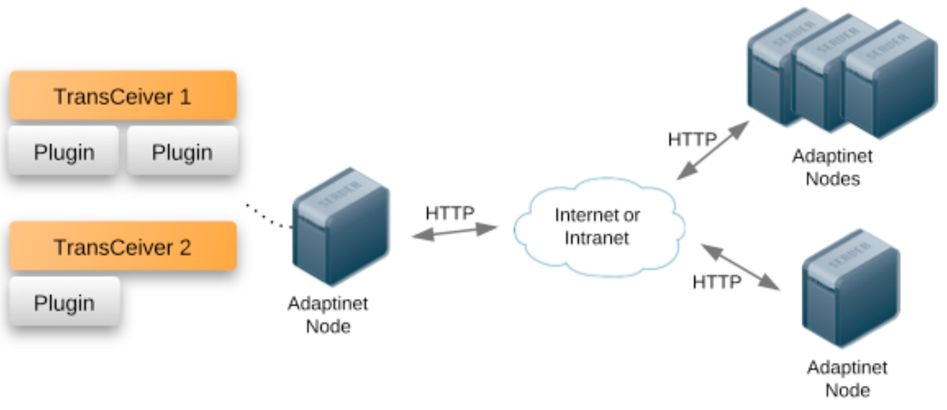
4.1 Design Options

Architecture #1, shown below is the simplest way of architecting Adaptinet Distributed Computing applications. A single plug-in is written, bundled and distributed with the TransCeiver. The plug-in contains all execution code for the Adaptinet distributed process.

Adaptinet nodes can reside within an Intranet or across the Internet. Or a combination of both.



Architecture #2 shown below is common if you need to take advantage of the additional processing capacity of each server. You can easily configure multiple Adaptinet TransCeivers to run on the same server by configuring different ports for each.



Note: Plug-ins have the ability to communicate with other plug-ins running on the same

TransCeiver by using the localPostMessage() method. See the JavaDocs for more specific details.

5 Configuration & Deployment

5.1 The Administration Console

The Administration Console is a web based GUI for configuring the Adaptinet Transceiver. You can use the console to change transceiver settings like port numbers, timeouts, SSL configuration, registering or de-registering plug-ins, modifying default peer files and much more.

How it Works

A TransCeiver must be running before the Administration Console can be used. The TransCeiver waits for requests from the Administration Console by listening on a port (the default port number is 44411). The port number can be changed by editing the transceiver.properties file (adminport=portnumber) or by changing the setting from within the Administration Console.

Running the Administration Console

The Administration Console can be run by simply addressing the IP and the admin port number of a local or remote TransCeiver. Start the local TransCeiver and wait for it to startup. Open a web browser and type in the default URL, http://localhost:44411. You are now running the Administrative Console for the local TransCeiver. From the Administration Console menu, you can make your selections.

The controls page

The only thing that you can do from this page is shutdown the TransCeiver. You might want to use this feature to shutdown remote TransCeivers. Keep in mind that once the TransCeiver is shutdown, the Administration console will no longer work.

The settings page

From the settings page you can tweak specific TransCeiver settings like the port numbers used, configuration file names and connection timeouts. Below is a table describing each property.

Setting	Description
showconsole	There are two types of consoles included with the Adaptinet Distributed Computing SDK. The Administration Console and the RunTime Console. This property refers to the RunTime Console. Change this setting to “false” to turn off the RunTime Console and “true” to turn it on. The RunTime Console is typically used for testing purposes
webroot	This property points to a directory for serving up web pages. You can use this feature to store HTML pages for browser based access to the TransCeiver. This is not an obvious feature, but you may find a use for it.
autoconnect	Set this property to “true” to enable auto connect capabilities. Typically, autoconnect should always be set to true.

autoreload	If this property is set to "true", when a plug-in is called from a remote TransCeiver, the local TransCeiver will check to see if the plug-in class has been modified and reload it if it has changed. NOTE: Always set this property to "false" before deploying your Adaptinet application. When set to "true" it greatly slows the execution of the plug-in.
connectiontimeout	From this property you can configure how long an actual socket connection will remain open before timing out. The default setting for this field is 120 seconds.
port	The port number for the TransCeiver.
peerfile	Points to the peerfile used by the TransCeiver. If a peerfile does not exist or is empty, the TransCeiver defaults to the defaultpeer.xml file. The peerfile and defaultpeer files are used for connecting to and conversing with the Adaptinet network.
classpath	Points to the classpath used to find the plug-ins.
httproot	Defines the directory in which the Administration Console HTML files and images are located.
verbose	Set to "true" or "false". When verbose mode is on ("true"), the TransCeiver will display a great deal of information regarding communication and internal execution.
pluginfile	Points to the XML file containing a list of Plug-ins registered with the TransCeiver.
adminport	The Administration Console port number.
secureport	If you would like to use SSL, set this property to the port number desired for SSL. The "port" property will be ignored if this property is set to any value other than 0.
sockettype	For internal use. Always set to "Plugin".
maxconnections	The maximum number of simultaneous connections that can be made to the TransCeiver at one time.

The plug-ins page
From the plug-ins page you can register new plug-ins or modify and/or delete existing plug-ins. To register a new plug, click the Add Plug-in button and fill in the plug-in related information.

Field	**Description**
Name	The name of the plug-in. The TransCeiver uses this name when it receives a message addressed to a plug-in.
Description	Could be anything. For your use only.
Class	This is the actual class that the TransCeiver will attempt to load (e.g. helloworld.Helloworld).

Classpath	This is the classpath that will be used for this plug-in.
Preload	This should be set to 1 if you would like the plug-in preloaded when the TransCeiver is started. Typically, all of the plug-ins that collectively make up an Adaptinet application should be set for preload. If preload is 0, the plug-in will not be loaded until it is called by a remote TransCeiver.

The peers page

The peers page points to your defaultpeers.xml file located in the ./Adaptinet directory. The default peers file is used for connecting to an Adaptinet Distributed Computing network. For more information about this topic, see the Peer Files section 5.3 in this guide.

The status page

The status page tells you what plug-ins are currently active for the TransCeiver.

The password page

Use this page to change the password for the Administrative Console. The default password is blank.

5.2 Configuring Your Application for Distribution

Now that you've built this powerful distributed application, you are ready for deployment. Follow the steps below:

1. The first thing that you should do is configure a TransCeiver with the plug-ins required for your application. Use the *Administration Console* to configure your plug-ins, class paths and default TransCeiver settings.
2. The next thing that you will need to do is to configure a default peer file for your application. You can use the *Administration Console* for this as well. See the Section 5.3 below for more detail about default peer files.
3. Configure and your application on the machines listed in your default peer file.
4. Now we are ready for deployment. You can package your application and distribute it. Make sure that the peers.xml file does not contain any peers. This will ensure that remote TransCeivers, when loaded for the first time, will use the default peer file to attach to the network.

Files Required by the TransCeiver for Deployment

The following files are required by the TransCeiver:

- Adaptinet.jar
- Peers.xml
- Defaultpeers.xml
- Properties.xml
- A Java Runtime Environment (JRE) 5 or greater

5.3 Peer Files

Every Adaptinet Distributed Computing application will run within its own Adaptinet Distributed Computing Network. It's easy to setup an Adaptinet Distributed Computing Network for your application by using the default peer file. How does the TransCeiver use the default peer file?

When a TransCeiver is loaded, it first looks at its peer file (peers.xml). Don't worry about the peer file right now, we'll cover this later. Let's assume that this is the first time that the TransCeiver is loading, its peer file should be empty (if you deployed your application properly). When the TransCeiver finds an empty peer file, it then looks at the default peer file (defaultpeers.xml).

The default peer file is provided by the author of the application and typically does not change after the application has been deployed. The default peer file should contain a peer or list of peers that have a very high likelihood of being up (configured and running the same application as the remote peer). These peers act as "seeds" for the Adaptinet Distributed Computing application and its network.

Connecting to the Network

When the TransCeiver is started it attempts to connect to the Adaptinet Distributed Computing Network by communicating with remote TransCeivers in the peer file (or default peer file if this is the first time). The TransCeiver receiving the connect message either connects the remote TransCeiver to the network or passes the request off to one of its peers (depending on its internal load).

When the TransCeiver is ultimately connected to the network, the remote TransCeiver connecting the local one passes back a peer file that now represents the local TransCeivers' immediate peers. This is eventually saved as the local peer file or peers.xml.

5.4 TransCeiver Command Line

The command line options for the TransCeiver are identical to the settings within the "settings page" on the Administrative Console (with exception to the –config option). See the *Administration Console* section for more details.

The –config option defines the properties file that is used by the TransCeiver. A typical command line is below:

```
java –cp ./adaptinet.jar com.adaptinet.transceiver.Transceiver
        -config transceiver.properties
```

6 Adaptive Characteristics of the TransCeiver

6.1 Overview

At Adaptinet we have created a decentralized system environment for distributed computing, translated this means that each participant in the system runs a piece of software that enables it to connect into a network of others running the same software. There is no upper limit on participation, and thus amazing scale can result in a relatively short time period.

6.2 The Use of Network Agents

To aid in the understanding of the frameworks' adaptive nature, we will examine in detail the characteristics and behaviors of the network agent. An agent resides on each node in the network and its primary responsibilities are to:

- Connect to the network
- Disconnect from the network
- Maintain the network connection
- Optimize the node's performance

To accomplish these tasks the agent contains a data structure that represents a local segment of the networks topology. If you traverse the network from node to node you would see a sliding window of the entire network topology. This alleviates the need for a central server that would contain a database about the network. If a node requires information about the network it will probe the network for the data.
In essence the network itself is the equivalent of the central server, it is not a static structure of connections but an entangled set of intelligent agents that are in constant communications.

6.3 Connecting and Disconnecting

It is important to understand the details of entering the network. Any node on the network can act as a gateway and support connecting other nodes to the network. When the transceiver starts part of its initialization is to activate the network agent. The agent will load the node file and begin probing for active nodes. If no active nodes are found, it will revert back to its default file. The default file should be established by the applications provider and should contain a set of seed nodes. If there is no default file or no node is available the agent will continue to probe the network for other nodes. The agent is also able to act as a gateway for other nodes to enter the network.

As important as entering the network properly exiting can be as critical. Once a node decides to disconnect it must inform the network of its exiting. First a substitute peer is selected from the closest neighbors. This node is given the network segment of the exiting node and the rest of the neighbors are informed of this change. This change is proliferated through the known local topology. Since the substitute node has taken on the exiting nodes topology it must pass on its topology to substitute. This process continues down through to the end of this segment.

6.4 Network Maintenance

Agents are also responsible for the maintenance of the network. At regular intervals an agent will probe the local segment to ensure its immediate peers are still available. If a peer does not respond to this request the agent will immediately begin to repair the network. This will guarantee the integrity of the local topology. If the each local topology remains stable the network overall will maintain stability not achievable by a centralized network.

In addition the time required to respond to these probes is recorded. This timing information can be later used to optimize the performance of the network. Slower peers can be shed as faster ones request a connection. If at any point the agent detects a condition it does not think it can rectify it will disconnect from this point in the network and attempt to reestablish a connection.

7 AdaptX – The TransCeiver Protocol

7.1 Overview

A developer using the Adaptinet Distributed Computing SDK doesn't need to be familiar with the AdaptX protocol because it is hidden by the Distributed Computing SDK. However, this section is provided to give you a better understanding of how the Adaptinet TransCeiver communicates with other TransCeivers on an Adaptinet Distributed Computing Network.

Adaptinet TransCeivers communicate with one another by posting XML messages to each other using HTTP. The AdaptX protocol describes in detail, where the message is coming from, its destination and its delivery characteristics including the plug-in name, method name and parameter content.

7.2 Communicating with a TransCeiver from Foreign Applications

Typically, Adaptinet TransCeivers communicate only with other Adaptinet TransCeivers. However, because the protocol of communication is an XML based protocol, foreign applications can easily communicate with TransCeivers and the Adaptinet network.

As discussed, an Adaptinet TransCeiver adheres to the AdaptX protocol, which is an XML based protocol. Foreign applications can talk to an Adaptinet Transceiver by posting XML messages to the TransCeiver via HTTP. The messages posted must adhere to the AdaptX protocol, which has been defined in an XML schema file named ADAPTX.XSD. This file is included with the Adaptinet Distributed Computing SDK.

7.3 A Sample AdaptX Message

The HelloWorld sample plug-in included with the Distributed Computing SDK generates the following AdaptX message when sending a hello message to itself. If it were communicating with a remote TransCeiver, the only difference would be designation of the <To> address.

```
<?xml version="1.0"?>
<Envelope>
   <Header>
        <Message>
          <To>
             <Address>
                <Prefix>http</Prefix>
                <Host>localhost</Host>
                <Port>8082</Port>
                <Plugin>HelloWorld</Plugin>
                <Method>Hello</Method>
             </Address>
          </To>
          <Timestamp>986641880918</Timestamp>
          <ReplyTo>
             <Address>
                <Prefix>http</Prefix>
                <Host>192.168.1.6</Host>
                <Port>8082</Port>
             </Address>
          </ReplyTo>
        </Message>
```

```
    </Header>
    <Body>
        <string>Hello, my name is 192.168.1.6:8082.</string>
    </Body>
</Envelope>
```

As you can see, the message format is encapsulated by an <Envelope> containing a <Header> and <Body>. The <Header> is primarily used for the definition of the sender and receiver. The <Body> is used to encapsulate the parameters of the method to be called on the remote TransCeiver.

7.4 AdaptX Details

The AdaptX protocol Aggregates are described in detail in the following section. Aggregates are XML tags that are used to encapsulate a collection of other XML tags.

The Envelope Aggregate

Tag	Required	Description
Header	Yes	The Header is currently used to encapsulate the Message aggregate. Future versions of the AdaptX protocol may include other aggregates within the Header.
Body	Yes	The Body of the message encapsulates the parameters of the method to be called on the remote TransCeiver.

The Header Aggregate

Tag	Required	Description
Message	Yes	The Message aggregate is used to encapsulate information about the source and destination of the message. Time stamping information is also included.

The Message Aggregate

Tag	Required	Description
To	Yes	This aggregate defines the address (or addresses) that the message will be delivered to.
Timestamp	Yes	When the TransCeiver sends a message, it will get the current time and set this tag to the number of seconds elapsed since midnight 1/1/1970.
ReplyTo	No	This aggregate defines the address that should be used by receiving TransCeivers as the “reply to” address. If a TransCeiver does not want to handle a message, it can simply pass the message on to another TransCeiver for processing also forwarding the <ReplyTo> aggregate so that a response (if any) can be returned to the original sender.

The Address Aggregate

Tag	Required	Description

Prefix	Yes	This element defines the protocol. http or https are valid values. Setting this element to https will cause the TransCeiver to use SSL will be used when sending the message.
Host	Yes	This element defines the host name or IP address.
Port	No	This element defines the port number upon which the TransCeiver is listening. The default value is 8082.
Plugin	Yes	The <Plugin> element corresponds to the Adaptinet Plug-in to be used or called.
Method	Yes	The <Method> element defines the Method to be called within the plug-in.

The Body Aggregate

Tag	**Required**	**Description**
Object	No	The <body> aggregate holds the parameters of the method to be called. This aggregate can hold a variety of tags including: <Array> <string> <boolean> <float> <double> <decimal> <integer> <long> <int> <short> <byte> <date> <Struct> When a TransCeiver receives the message, it converts the parameters to their corresponding type and delivers them to the method within the plug-in.