Niche Quick Start Guide

Leif Lindbäck and Vladimir Vlassov

Royal Institute of Technology (KTH), Stockholm, Sweden
{leifl,vladv}@kth.se

Table of Contents

Introduction

Niche is a Distributed Component Management System (DCMS), which is used to develop, deploy and execute self-managing distributed component-based applications on a structured overlay network of computers. Niche includes (1) a set of APIs for the development of self-managing distributed applications; (2) a run-time execution environment for the deployment and execution of the applications together with its management elements. The Niche run-time environment includes a set of containers that reside on a structured overlay network of computers; and a set of the overlay services (resource discovery, deployment, publish-subscribe, metadata DHTs) provided in each of the containers.

The main innovation in Niche is the novel use of overlays, where a structured overlay provides a network-transparent sensing/actuation infrastructure that enables self-management of distributed component-based applications. Functional components, component groups and bindings are first-class entities in Niche that can be monitored and manipulated by management components (via sensors and actuators) using the extensible monitoring and actuation APIs of Niche. Management components are organized as a network of Management Elements interacting through events. Niche supports sensing changes in the state of components and environment, and allows individual components to be found and appropriately manipulated. Niche deploys both functional and management components and sets up the appropriate sensor and actuation support infrastructure. The initial deployment code can be either manually written by the developer, or generated by Niche from an ADL (Architecture Description Language) description of the application architecture. The ADL compiler for describing initial configurations is made available together with Niche.

1 Downloading Niche

Download and unpack the `niche-0.2.zip` archive from `http://niche.sics.se/`

2 Installing Niche

1. In order to be deployed and to operate, Niche requires the Apache Ant build tool to be installed. For Ant download and installation, see `http://ant.apache.org/index.html`.
2. Niche requires Java (`http://java.sun.com`), version 1.6 or higher.
3. In the file `niche-0.2/Jade/etc/dks/dksParam.prop`, substitute `localhost` for your real host (host could be given as either host name or ip address) in the entry `ip`. Note that you can not run Niche without a working network connection.
4. In the file `niche-0.2/Jade/etc/oscar/bundle-jadeboot.properties`, substitute `localhost` for your real host in the entries `jadeboot.registry.host` and `jadeboot.discovery.host`.

5. In the file `niche-0.2/Jade/etc/oscar/bundle-jadenode.properties`, substitute `localhost` for your real host in the entries `jadeboot.registry.host` and `jadeboot.discovery.host`.
6. Make sure your host name is associated with your correct host. On Linux/UNIX machines this is done the following way:
   (a) Find the host name with the command `uname -n`
   ```
   $ uname -n
   myHostName
   ```
   (b) Make sure there is a line in the `/etc/hosts` file that looks like `<ip address> <host name>`, where `<host name>` is the host name returned by the `uname` command in the previous step.
7. Change all occurrences of `<Path to Jade>` to the absolute path to the `niche-0.2/Jade` directory in the following files:
   - `niche-0.2/Jade/etc/execute.properties`
   - `niche-0.2/Jade/etc/oscar/bundle.properties`
   - `niche-0.2/Jade/etc/oscar/bundle-jadeboot.properties`
8. Copy the version of `niche-0.2/Jade/externals/swt-3349-*.jar`, where * corresponds to the name of your operating system, e.g windows, to `niche-0.2/Jade/externals/swt.jar`
9. Niche uses a web server to provide addresses of existing nodes to new joining nodes. The web server must be installed separately since it is not part of the Niche distribution. The only requirement on the web server is that it must support PHP. When you have a working web server, do the following:
   (a) Copy the directory `niche-0.2/Jade/webcache` to the www root of the web server.
   (b) In `niche-0.2/Jade/etc/dks/dksParam.prop`, specify the host and port number of the web server in the entry `publishAddress`. Note that you can not specify `localhost`, you must use a real host.

3 Starting Niche

1. Create run-time archives by running Ant with the target `bundles` specified in the build file `niche-0.2/Jade/build-src.xml`, as follows.
   ```
   cd niche-0.2/Jade/
   ant -f build-src.xml bundles
   ```
2. Start the first node by running Ant with the target `jadeboot` in the build file `niche-0.2/Jade/build.xml`, as follows.
   ```
   cd niche-0.2/Jade/
   ant jadeboot
   ```
3. Start more nodes by by running Ant with the target `jadenode` in the build file `niche-0.2/Jade/build.xml`, as follows. You can start nodes on the same or different machines. The following commands should be executed for each node you want to start.
   ```
   cd niche-0.2/Jade/
   ant jadenode
   ```

4 Defining Niche Components with Fractal

Niche uses the *Fractal Architecture Description Language (ADL)* to define components. For more information about Fractal and ADL see the Niche Programming Guide or the Fractal home page (`http://fractal.ow2.org/`).

Briefly, Fractal components are runtime entities that communicate exclusively through well-defined access points, called *interfaces*. In Niche the interfaces are regular Java interfaces. Interfaces are divided into two kinds: *client interfaces* that emit operation invocations and *server interfaces* that receive them. Interfaces are connected through communication paths, called *bindings*. Each Fractal component has (at least) two basic controllers, (1) *binding controller*, which supports binding and unbinding; (2) *life-cycle controller* supports starting and stopping the execution of the component.

5 The Niche Hello World Program

Fig. 1. The Hello World program's GUI.

The Hello World program has front-end components and service components. The number of components is configurable. By default there are three service components and one front-end component. The service components are placed in a component group. The front-end component displays the GUI showed in Figure 1. There is a one-to-any binding from the front-end to the service component group, which is invoked when the *Hello Any* button in the GUI is clicked. When this happens, *one* of the service components will print the string `Hello World`. There is also a one-to-all binding from the front-end to the service component group, which is invoked when the *Hello All* button in the GUI is clicked. In this case *all* service components will print the string `Hello World`.

The Hello World program is self-healing, which means that if one service component crashes a new component will be instantiated, deployed and started. The self-healing control loop consists of two Management Elements (MEs), the `ServiceSupervisor` aggregator and the `ConfigurationManager` manager. `ServiceSupervisor` is notified whenever a service component leaves or joins the service component group. It notifies `ConfigurationManager` if the number of components in the group drops below 3. When notified, `ConfigurationManager` deploys and starts a new service component.

```
1   <component name="frontend">
2       <interface name="helloAny"
3                  role="client
4                  signature="helloworld.interfaces.HelloAnyInterface"
5                  contingency="optional"
6       />
7       <interface name="helloAll"
8                  role="client"
9                  signature="helloworld.interfaces.HelloAllInterface"
10                 contingency="optional"
11      />
12      <content class="helloworld.frontend.FrontendComponent"/>
13      <virtual-node name="lightweight1" resourceReqs="10"/>
14  </component>
```

Listing 1. ADL definition of front-end component.

The Hello World program also includes `StartManager`, which instantiates and deploys the service component group, `ServiceSupervisor` and `ConfigurationManager`. This is necessary since the current Niche prototype does not allow defining groups or configuring MEs in ADL.

The Hello World program is included in the Niche distribution, in the directory `NicheHelloWorld`; it is explained in detail below.

5.1 Functional Components of Hello World

There are two types of functional components, front-end component and service component.

Front-End Component The front-end component has two tasks: (1) to accept input from the user through a GUI; (2) call service component methods in response to user commands entered through the GUI.

The GUI is in a separate class, `helloworld.frontend.UserInterface`, which is not explained here. For an introduction to writing user interfaces with Swing, see `http://java.sun.com/docs/books/tutorial/uiswing/`. This class is not aware of Niche, its only task is to handle the GUI.

The frontend component is defined in the ADL file, `NicheHelloWorld.fractal`, located in the root directory of the Hello World program (`niche-0.2/NicheHelloWorld`). The component definition is showed in Listing 1. Lines 2-6 define the interface used for the one-to-any binding explained in 5. Note that the role of the interface is `client` (line 3), which means that the front-end component invokes this interface. Line 4 specifies the Java interface that defines the operations in the `helloAny` interface. Lines 7-11 specifies the interface used for the one-to-all binding explained in 5. Line 12 specifies the main Java class of the component. This class must implement server interfaces of the component, and also implement the interfaces of the Fractal controllers, see 4. Line 13 defines (1) a logical name for the Niche node on which the front-end component will be deployed; (2) a requirement for the node. This

requirement has no meaning, it is just a number. It is matched against the available resource specified for each node in the file `niche-0.2/Jade/lines`. Each line in this file contains a node number and a resource. A component specified in `NicheHelloWorld.fractal` will only be deployed on nodes with resource value (in `lines`) at least the value specified in the requirement for that node (i.e greater than or equal to 10 in this case). In a real application this feature could be used to specify for example available memory or CPU speed. This is explained in the Niche Programming Guide.

As specified in the ADL definition in Listing 1, it is the class `helloworld.frontend.FrontendComponent` that implements the Fractal controller interfaces. This part of the class is shown in Listing 2. The `listFc` method (lines 1-3) returns an array with the names of all components to which the front-end component can be bound to. These are the components implementing the front-end's client interfaces and `component`, which is a reference to the front-end component itself. All components should be able to handle such a reference to itself. The `lookupFc` method (lines 5-15) returns the front-end's current reference to the specified component. The `bindFc` method (lines 17-27) stores the given reference to a component. The `unbindFc` method (lines 29-39) removes the reference to the specified component. The `getFcState` method (lines 41-43) tells whether the front-end is active or not, `status` is a boolean instance parameter not showed in the listing. `startFc` (lines 45-50) is called by Niche when the front-end should become active, line 47 instantiates the `UserInterface` class containing the GUI. This is when the GUI is displayed. Finally, `stopFc` (lines 52-54) is called by Niche when the front-end should become inactive.

Listing 3 shows the methods that are invoked when the user clicks one of the GUI buttons (`HelloAny` or `HelloAll`). The component bound to the appropriate client interface is invoked. When the user clicks `HelloAny`, the GUI event handler will invoke the `helloAny` method defined in lines 5-7 in Listing 3. This method in turn invokes the method `helloAny` in the component who's reference is stored in the `helloAny` instance parameter as shown in line 6. The correct reference is stored in this instance parameter by the `StartManager`, as explained in 5.2. `HelloAll` is handled in a similar way as `HelloAny`.

Service Component The service component's task is to print the specified string when any of its server interfaces is invoked.

The service component consists of one class who's source code is in `src/helloworld/service/ServiceComponent.java`. Listing 4 shows its implementation of the server interfaces (`helloAny` and `helloAll`). These methods print the specified string.

The ADL definition is in the same fractal file as the front-end component's definition, `NicheHelloWorld.fractal`, which is located in the root directory of the Hello World program (`niche-0.2/NicheHelloWorld`). Note that there must be one `component` entity for each service component that shall be started. Since there are three such entities there will be three instances of the service component.

```
1     public String[] listFc() {
2         return new String[] { "component", "helloAny", "helloAll" };
3     }
4
5     public Object lookupFc(final String itfName) throws
          NoSuchInterfaceException {
6         if (itfName.equals("helloAny")) {
7             return helloAny;
8         } else if (itfName.equals("helloAll")) {
9             return helloAll;
10        } else if (itfName.equals("component")) {
11            return myself;
12        } else {
13            throw new NoSuchInterfaceException(itfName);
14        }
15    }
16
17    public void bindFc(final String itfName, final Object itfValue)
          throws NoSuchInterfaceException {
18        if (itfName.equals("helloAny")) {
19            helloAny = (HelloAnyInterface) itfValue;
20        } else if (itfName.equals("helloAll")) {
21            helloAll = (HelloAllInterface) itfValue;
22        } else if (itfName.equals("component")) {
23            myself = (Component) itfValue;
24        } else {
25            throw new NoSuchInterfaceException(itfName);
26        }
27    }
28
29    public void unbindFc(final String itfName) throws
          NoSuchInterfaceException {
30        if (itfName.equals("helloAny")) {
31            helloAny = null;
32        } else if (itfName.equals("helloAll")) {
33            helloAll = null;
34        } else if (itfName.equals("component")) {
35            myself = null;
36        } else {
37            throw new NoSuchInterfaceException(itfName);
38        }
39    }
40
41    public String getFcState() {
42        return status ? "STARTED" : "STOPPED";
43    }
44
45    public void startFc() throws IllegalLifeCycleException {
46        // Create the GUI.
47        new UserInterface(this);
48        status = true;
49        System.err.println("Frontend Started.");
50    }
51
52    public void stopFc() throws IllegalLifeCycleException {
53        status = false;
54    }
```

Listing 2. Implementation of Fractal controllers in front-end component.

```
1   // ////////////////////////////////////////////////////////////
2   // ////////////////// Called from the user interface. //////////////
3   // ////////////////////////////////////////////////////////////
4
5   public synchronized void helloAny() {
6       helloAny.helloAny("HelloWorld");
7   }
8
9   public synchronized void helloAll() {
10      helloAll.helloAll("HelloWorld");
11  }
```

Listing 3. Calls from GUI in front-end component.

```
1   public void helloAny(String s) {
2       System.out.println(s);
3   }
4
5   public void helloAll(String s) {
6       System.out.println(s);
7   }
```

Listing 4. Implementations of server interfaces in service component.

The component's definition and implementation of Fractal controllers are very similar to the front-end component's and are therefore not shown.

5.2 Management Elements of Hello World

Management elements (ME) can be defined in ADL in a Fractal file that should have the same name as the Java class implementing the element, and be located in the same directory. Such a definition will not cause the deployment of the ME, it must be deployed programmatically.

The MEs are:

- `ServiceSupervisor`, which is notified whenever a service component joins or leaves the service component group. If the number of components in this group drops below three it notifies `ConfigurationManager`
- `ConfigurationManager`, which deploys a new service component, starts it and makes it join the service component group.
- `StartManager`, which instantiates and deploys the service component group, `ServiceSupervisor` and `ConfigurationManager`. This is necessary since the current prototype does not allow defining groups or configuring MEs in ADL.

Management elements are described in detail below.

ServiceSupervisor. The `ServiceSupervisor` is an aggregator, which is defined in `ServiceSupervisor.fractal`, shown is Listing 5. Lines 2-4 define the following server interfaces implemented by this ME.

```
<definition name="helloworld.aggregators.ServiceSupervisor" >
 <interface name="init" role="server"
      signature="dks.niche.fractal.interfaces.InitInterface" />
 <interface name="eventHandler" role="server"
      signature="dks.niche.fractal.interfaces.EventHandlerInterface" />
 <interface name="movable" role="server"
      signature="dks.niche.fractal.interfaces.MovableInterface" />
 <interface name="trigger" role="client"
      signature="dks.niche.fractal.interfaces.TriggerInterface" />
 <content class="helloworld.aggregators.ServiceSupervisor" />
</definition>
```

Listing 5. ServiceSupervisor definition.

```
String idAsString =
    failedEvent.getFailedComponentId().getId().toString();

if (!currentComponents.containsKey(idAsString)) {
    // The failed component was not in our list of
    // active service components.
    return;
}

// Remove failed component from list of active components.
currentComponents.remove(idAsString);
currentAllocatedServiceComponents--;

if (myId.getReplicaNumber() < 1) {
    System.out.print("ServiceSupervisor ");
} else {
    System.out.print("ServiceSupervisor REPLICA ");
}
System.out.println("has received a ComponentFailEvent!\n"
    + "The result is that there are now " +
        currentAllocatedServiceComponents
    + " service components");

if (currentAllocatedServiceComponents <
    MINIMUM_ALLOCATED_SERVICE_COMPONENTS) {
    // There are too few service components.
    if (myId.getReplicaNumber() < 1) {
        System.out.println("ServiceSupervisor triggering!");
    }

    // Cancel eventual running timers since we are telling
    // ConfigurationManager now.
    actuator.cancelTimer(availabilityTimerId);
     // Tell ConfigurationManager there are too
     // few service components.
    eventTrigger.trigger(new ServiceAvailabilityChangeEvent());

    // When the timer goes off we will check if enough service
    // components have become active.
    availabilityTimerId =
        actuator.registerTimer(this,
                    AvailabilityTimerTimeoutEvent.class,
                    CHECK_NR_OF_COMPONENTS_AFTER_THIS_TIME);
}
```

Listing 6. Handling of ComponentFailEvent in ServiceSupervisor.

- `InitInterface`, which contains (1) method for receiving a reference to a component implementing the Niche API; (2) methods for receiving initialization attributes specified at deploy time.
- `EventHandlerInterface`, which lets the ME receive events.
- `MovableInterface`, which enables reading the ME's state when the ME is copied or moved.

Line 5 defines the client interface, `TriggerInterface`, to which this ME should be bound. This interface is used to send events, in this case to `ConfigurationManager`. A reference to a component implementing this interface will automatically be handed to `ServiceSupervisor` when it is deployed. Finally, line 6 defines the implementing class, `ServiceSupervisor`. Listing 6 shows what happens when `ServiceSupervisor` receives a `ComponentFailEvent`: it sends an event to `ConfigurationManager` if there are less than three active service components. Comments in the listing explain how it works.

ConfigurationManager. The `ConfigurationManager` is a manager, which is defined in `ConfigurationManager.fractal`; it is very similar to the definition of `ServiceSupervisor`. Listing 7 shows what happens when `ConfigurationManager` receives a `ServiceAvailabilityChangedEvent`: it deploys and starts a new `ServiceComponent`. Comments in the listing explain how it works.

StartManager. The `StartManager` is a manager which is defined in `NicheHelloWorld.fractal`; the definition is shown in Listing 8. Note that the value of the `definition` attribute is `ManagementType` (line 1). This definition contains a set of client interfaces that correspond to the Niche API. These interfaces are automatically bound after `StartManager` is instantiated. Listing 9 show how the StartManager creates the service component groups; Listing 10 shows how the `ServiceSupervisor` and `ConfigurationManager` are configured and deployed. Both listings are explained by comments in the code.

5.3 Deploying Hello World

1. Place the file `NicheHelloWorld.fractal` in `niche-0.2/Jade/examples` and the file `nichehelloworld.jar` in `niche-0.2/Jade/externals`. This is already done in the downloadable Niche distribution.
2. The files listed below all need the correct properties concerning the Hello World program. In the distribution all needed properties are already defined.
   - `niche-0.2/Jade/build-src.xml`
   - `niche-0.2/Jade/etc/build.properties`
   - `niche-0.2/Jade/META-INF/jadenode/MANIFEST.MF`
   - `niche-0.2/Jade/META-INF/jadeboot/MANIFEST.MF`

```
1   // Find a node that meets the requirements for a service component.
2   NodeRef newNode = null;
3   try {
4       newNode = myManagementInterface.
5                       oneShotDiscoverResource(nodeRequirements);
6   } catch (OperationTimedOutException err) {
7       System.out.println("Discover operation timed out.");
8       continue;
9   }
10  if (newNode == null) {
11      System.out.println("Could not get a new resource.);
12      break;
13  }
14  System.out.println("Got a resource");
15
16  // Allocate resources for a service component at the found node.
17  List allocatedResources = null;
18  try {
19      allocatedResources = myManagementInterface.allocate(newNode,
            null);
20  } catch (OperationTimedOutException err) {
21      System.out.println("Allocate operation timed out.");
22      continue;
23  }
24  ResourceRef allocatedResource = (ResourceRef)
        allocatedResources.get(0);
25
26  // Deploy a new service component instance at the allocated resource.
27  String deploymentParams = null;
28  try {
29      deploymentParams = Serialization.serialize(serviceCompProps);
30  } catch (IOException ioe) {
31      ioe.printStackTrace();
32  }
33  List deployedComponents = null;
34  try {
35      deployedComponents =
36          myManagementInterface.deploy(allocatedResource,
                deploymentParams);
37  } catch (OperationTimedOutException err) {
38      System.out.println("Deploy operation timed out.");
39      continue;
40  }
41
42  ComponentId cid =
43      (ComponentId) ((Object[]) deployedComponents.get(0))[1];
44  System.out.println("ConfigurationManager is adding new component");
45
46  // Add the newly deployed component to the service group
47  // and start it.
48  myManagementInterface.update(componentGroup, cid,
49          NicheComponentSupportInterface.ADD_TO_GROUP_AND_START);
50  System.out.println("ConfigurationManager says: All done!");
```

Listing 7. Handling of ServiceAvailabilityChangeEvent in ConfigurationManager.

```
1   <component name="StartManager"
        definition="org.objectweb.jasmine.jade.ManagementType">
2     <content class="helloworld.managers.StartManager"/>
3   </component>
```

Listing 8. StartManager definition.

```
1         // Get a reference to the Niche API.
2         NicheActuatorInterface myActuatorInterface =
              nicheService.getOverlay().getJadeSupport();
3
4         // Find the front-end component.
5         ComponentId frontendComponent =
6             (ComponentId) nicheIdRegistry.lookup(APPLICATION_PREFIX +
                  FRONTEND_COMPONENT);
7
8         // Find all service components.
9         ArrayList<ComponentId> serviceComponents = new ArrayList();
10        int serviceComponentIndex = 1;
11        ComponentId serviceComponent =
12            (ComponentId) nicheIdRegistry.lookup(APPLICATION_PREFIX +
                  SERVICE_COMPONENT
13                + serviceComponentIndex);
14        while (serviceComponent != null) {
15            serviceComponents.add(serviceComponent);
16            serviceComponentIndex++;
17            serviceComponent =
18                (ComponentId) nicheIdRegistry.lookup(APPLICATION_PREFIX
                      + SERVICE_COMPONENT
19                    + serviceComponentIndex);
20        }
21
22        // Create a component group containing all service components.
23        GroupId serviceGroupTemplate =
              myActuatorInterface.getGroupTemplate();
24        serviceGroupTemplate.addServerBinding("helloAny",
              JadeBindInterface.ONE_TO_ANY);
25        serviceGroupTemplate.addServerBinding("helloAll",
              JadeBindInterface.ONE_TO_MANY);
26        GroupId serviceGroup =
27            myActuatorInterface.createGroup(serviceGroupTemplate,
                  serviceComponents);
28
29        // Create a one-to-any binding from the front-end
30        // to the service group.
31        // This binding uses the helloAny interface.
32        String clientInterfaceName = "helloAny";
33        String serverInterfaceName = "helloAny";
34        myActuatorInterface.bind(frontendComponent,
              clientInterfaceName, serviceGroup,
35                                 serverInterfaceName,
                                       JadeBindInterface.ONE_TO_ANY);
36
37        // Create a one-to-all binding from the front-end
38        // to the service group.
39        // This binding uses the helloAll interface.
40        clientInterfaceName = "helloAll";
41        serverInterfaceName = "helloAll";
42        myActuatorInterface.bind(frontendComponent,
43                                 clientInterfaceName,
44                                 serviceGroup,
45                                 serverInterfaceName,
46                                 JadeBindInterface.ONE_TO_MANY);
```

Listing 9. Creation of service component groups in StartManager.

```
1   // Configure and deploy the ServiceSupervisor aggregator.
2   ManagementDeployParameters params =
3       new ManagementDeployParameters();
4   params.describeAggregator(ServiceSupervisor.class.getName(),
5                             "SA", null,
6                             new Serializable[] {
                                  serviceGroup.getId() });
7   params.setReliable(true);
8   NicheId serviceSupervisor =
9       myActuatorInterface.deployManagementElement(params,
            serviceGroup);
10  myActuatorInterface.subscribe(serviceGroup,
11                            serviceSupervisor,
12                            ComponentFailEvent.class.getName());
13  myActuatorInterface.subscribe(serviceGroup,
14                            serviceSupervisor,
15                            MemberAddedEvent.class.getName());
16
17  // Grab the service component's properties from a
18  // service component which is already deployed.
19  // The ConfigurationManager needs these when
20  // it deploys a new service component.
21  ComponentId grabParametersFromThis =
22      (ComponentId) nicheIdRegistry.lookup(APPLICATION_PREFIX +
            SERVICE_COMPONENT + "1");
23  DeploymentParams serviceComponentProperties = null;
24  try {
25      serviceComponentProperties =
26          (DeploymentParams)
                Serialization.deserialize(grabParametersFromThis.
27                                getSerializedDeployParameters());
28  } catch (IOException e) {
29      e.printStackTrace();
30  } catch (ClassNotFoundException e) {
31      e.printStackTrace();
32  }
33
34  // Configure and deploy the ConfigurationManager manager.
35  String minimumNodeCapacity = "200";
36  params = new ManagementDeployParameters();
37  params.describeManager(ConfigurationManager.class.getName(),
38                  "CM", null,
39                  new Serializable[] { serviceGroup,
40                                       serviceComponentProperties,
41                                       minimumNodeCapacity });
42  params.setReliable(true);
43  NicheId configurationManager =
44      myActuatorInterface.deployManagementElement(params,
            serviceGroup);
45  myActuatorInterface.subscribe(serviceSupervisor,
        configurationManager,
46      ServiceAvailabilityChangeEvent.class.getName());
```

Listing 10. Configuration and deployment of management elements in StartManager.

3. The file `niche-0.2/Jade/NicheHelloWorld-Deploy.bsh` is a script needed for deployment of Hello World.
4. Start Niche as described in Section 3. You should start at least four nodes including the first node started with the Ant target `jadeboot`.
5. Deploy the Hello World program by running Ant with the target `testG4A-NicheHelloWorld-Deploy`, as follows.
   ```
   cd niche-0.2/Jade/
   ant testG4A-NicheHelloWorld-Deploy
   ```

5.4 Starting Hello World

1. Start the Hello World program by running Ant with the target `testG4A-NicheHelloWorld-Start`, as follows.
   ```
   cd niche-0.2/Jade/
   ant testG4A-NicheHelloWorld-Start
   ```
   Note that this target needs the script file `niche-0.2/Jade/NicheHelloWorld-Start.bsh`.
2. When this target is executed, the GUI in Figure 1 will be displayed. How the program works is explained in the beginning of Section 5.