SDS developer guide

Develop distributed and parallel applications in Java

Nathanaël Cottin

sds@ncottin.net

http://sds.ncottin.net

version 0.0.3

Abstract

This guide is provided to help developers integrate *SDS* to build distributed and parallel applications as well as mobile agents.

Keywords: distributed, parallel, mobile, java

Contents

1 Introduction

Simple Distributed system (*SDS* – to be pronounced "*sudes*") is an open-source Java implementation of a network asynchronous messaging system.

It is designed taking scalability, efficiency and security considerations in concern (although version `1.0.0` defines a limited security implementation based on nonces).

SDS is primarily developped to build distributed and parallel systems and design mobile agents.

This guide refers to *SDS* version `1.0.0 beta 2`.

2 SDS overview

SDS is event-driven, which means that distributed objects' implementations can perform their tasks while receiving asynchronous messages.

SDS defines the following services, depicted by figure 1:

- A *manager* used to register distributed objects and collect their network references
- A *daemon* called by managers, takes distributed objects' life cycle management. Particularly, this service loads and activates distributed objects on its local host (i.e. distant host from managers' perspective) and allows to terminate managed distributed objects.

3 Daemons discovery

SDS provides two ways to discover daemons using one of a manager's `registerDaemons()` operations:

- Giving already known daemons references (not described here)
- Providing a URL, as described hereafter.

SDS managers are able to retrieve daemons references via URLs. A list of root URLs can be set. Each root URL points to a list of references' URLs. These references must be updated by daemons when launched.

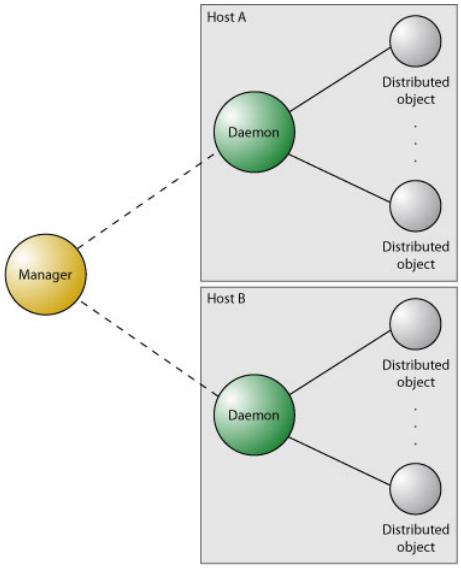

Figure 1: SDS architecture overview

3.1 Step 1: daemons references publication

Figure 2 indicates two daemons running on distant hosts (from the main program's perspective) publish their references (using `getReference()`) on downloadable URLs.

These locations (pointing to daemons references) are gathered together in a single URL (one reference URL per line) called *references list*. Thus this main URL returns two references URLs.

Daemons do not implement the *singleton* design principle. Two or more daemons can run on a single host.

3.2 Step 2: daemons references list access

The main program instantiates a manager which consults the references list URL (figure 3). At this stage, the manager knows that two daemons are supposed to be running and owns the URLs to read their references.

3.3 Step 3: daemons references access

The manager iterates through the references list URLs to obtain the dameons references (figure 4).

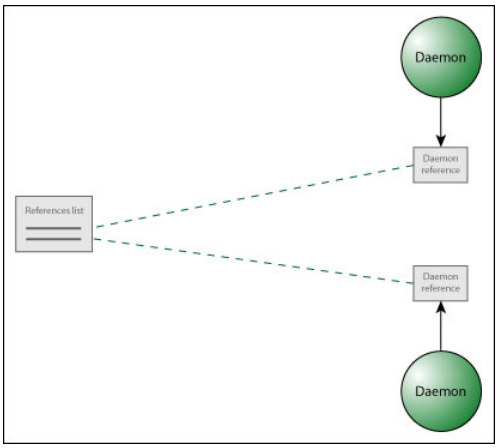

Figure 2: Daemons discovery – step 1

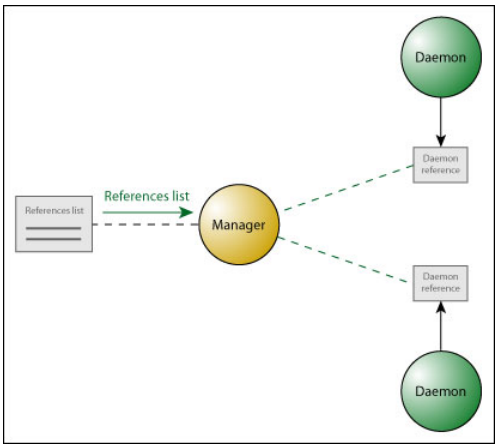

Figure 3: Daemons discovery – step 2

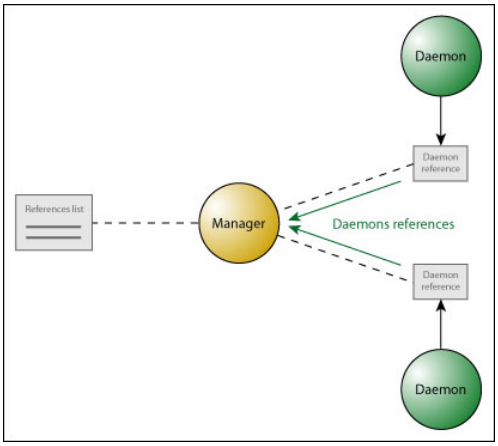

Figure 4: Daemons discovery – step 3

3.4 Step 4: daemons references storage

The manager locally stores the collected daemons references (adding to previously registered daemons references) and returns a copy back to the caller.

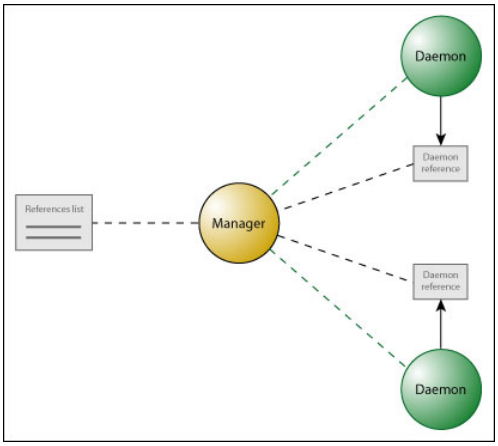

Figure 5: Daemons discovery – step 4

4 Message delivery steps

Asynchronous and non-blocking message delivery is one of the core concepts of *SDS*, though it also allows to block until a well-identified message is received.

4.1 Step 1: required architectural components

A manager (from `sds.services` package) has knowledge of a demon's network reference (this is depicted by a dashed line on figure 6). The manager and the daemon may reside on different hosts, as far as the latter are visible (`ping` succeeds).

Furthermore, the manager has read access to a distributed object's bytecode (`.class` file).

The daemon currently waits for interactions with a manager. No distributed object is running the daemon's host.

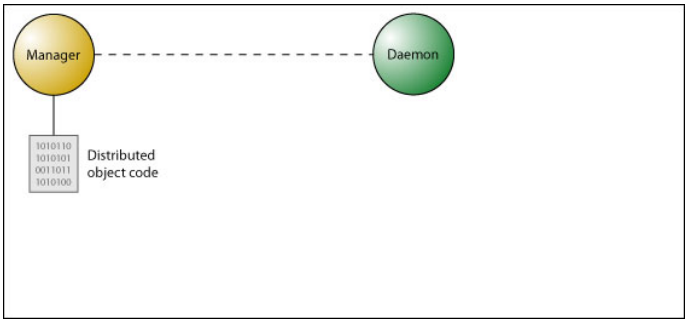

Figure 6: Message delivery – step 1

4.2 Step 2: distributed object delivery

When the manager is asked to register a distributed object on one of its known daemons, it creates a message containing the distributed object's bytecode and sends it to the specified daemon (figure 7). This message include transactional information as the manager expects a response from the daemon (in step 4).

4.3 Step 3: distributed object construction

As depicted by figure 8, the daemon receives a registration request from the manager. It builds the distributed object class and generates an instance. If the distributed object runs as a server, the daemon starts this object on a selected port number.

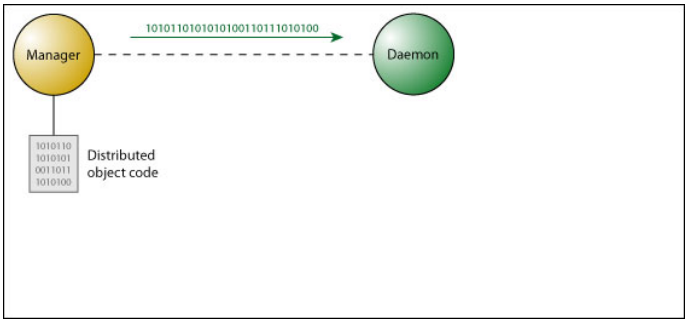

Figure 7: Message delivery – step 2

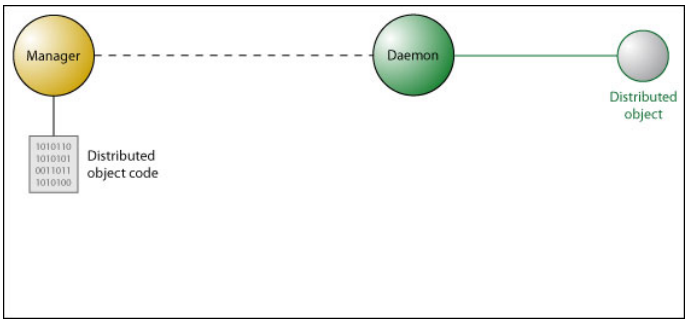

Figure 8: Message delivery – step 3

4.4 Step 4: reference callback

The daemon sends a reply to the manager to indicate the activated distributed object's reference (figure 9).

The manager, which was waiting for this answer, gives this reference in return of the registration process.

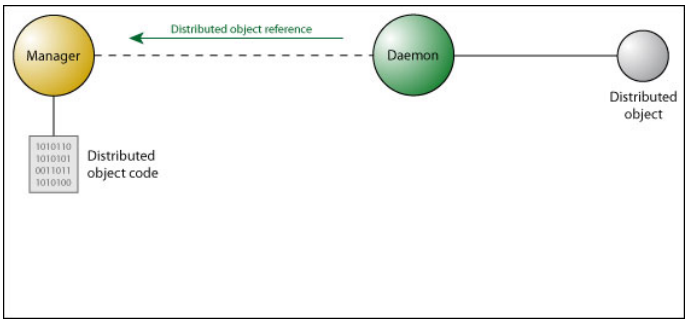

Figure 9: Message delivery – step 4

4.5 Step 5: recipient reference knowledge

Another distributed object able to access the manager's registration result can read this reference (figure 10) to deliver a message to the newly (distant) created distributed object.

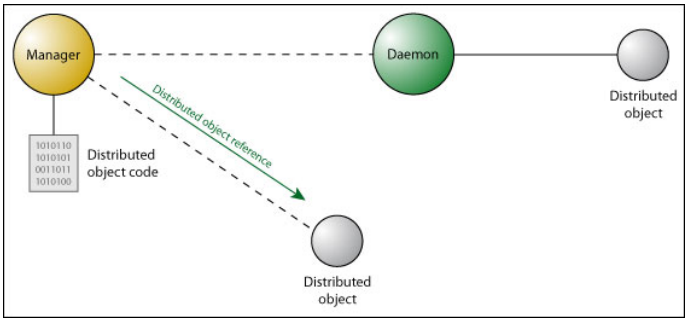

Figure 10: Message delivery – step 5

Note that each distributed object activated by a daemon holds its source manager reference.

4.6 Step 6: distributed object call

The distant distributed object is dynamically invoked as far as the message sender provides a valid recipient reference in the transmitted message (figure 11).

As all messages deliveries are asynchronous, this message may enclose transactional information if a reply is needed.

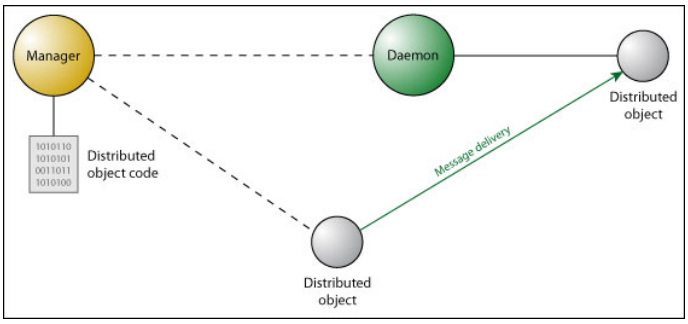

Figure 11: Message delivery – step 6

5 SDS security implementation

As is stands, *SDS* implements security by means of nonces and does not include cryptographic operations on messages. This will be part of a future release.

6 Transactional exchanges

Although *SDS* is asynchronous, it provides the ability to set transactional identifiers to request messages and wait for adequate returned messages. The waiting process can be either synchronous (blocking) or asynchronous (main tasks keep running).

7 Examples

7.1 "Feedback" distributed algorithm

7.1.1 Description

This distributed algorithm aims at transmitting an information to a graph. A special node, called *initiator* receives a start message from the environment (i.e.

main program). The feedback of all other nodes allowes the initiator to inform the environment about transmission process termination.

The following tags are used:

- `NEIGHBOURS_TAG`: used by the manager to send direct neighbours references to each node.
- `INIT_TAG`: starts the distributed algorithm on a single node (the initiator). This message contains the information to transmit to all nodes.
- `MESSAGE_TAG`: indicates the information reception (sent by another node).
- `RETURN_TAG`: when a node has delivered the information to all direct neighbours, it sends this this tag to its parent node (i.e. the node which first sent the information).
- `END_TAG`: used by the initiator to notify the manager that the information transmission process (the algorithm) is finished.

The "main" operation of this example runs a daemon and creates a feedback manager which connects to the daemon to send 3 feedback servers. This is an all-to-one example and real world applications require to run daemons and managers separately.

The source code is available on *SDS* website (in the `examples` section).

7.1.2 Message tags source code

```
1  /**
    * Copyright 2007 Nathanael Cottin
    */
   package test;

6  /**
    * @author ncottin
    */
   public final class FeedbackTags {

11   public static final int NEIGHBOURS_TAG = 0;
     public static final int INIT_TAG   = 1;
     public static final int MESSAGE_TAG  = 2;
     public static final int RETURN_TAG  = 3;
     public static final int END_TAG    = 4;
16 }
```

7.1.3 Algorithm implementation source code

```
1  /**
    * Copyright 2007 Nathanael Cottin
    */
4  package test;

   import java.util.Collection;

   import sds.DistributedServerObject;
9  import sds.data.Message;
   import sds.data.Reference;
   import sds.error.SdsException;

   public final class FeedbackServer extends DistributedServerObject {
14
    private Message      initMsg    = null;
    private Collection<Reference> remainingNeighbours = null;
    private Reference    parent     = null;
    private String       value      = null;
19
    public FeedbackServer() {
     super(false);
    }

24  @Override
    public void execute() {
    }

    @Override
29  public void processError(SdsException error) {
     System.out.println("Feedback server: " + getReference());
     error.printStackTrace();
    }

34  @SuppressWarnings("unchecked")
    @Override
    public boolean receive(Message msg) throws SdsException {
     switch (msg.getTag()) {
      case FeedbackTags.NEIGHBOURS_TAG: {
39     Collection<Reference> n = (Collection<Reference>) msg.getData();
       if (n != null) {
        remainingNeighbours = n;
       }

44     Reference ref = getReference();
```

```
        System.out.println(ref + " - received " + n.size()
         + " neighbour(s):");
        for (Reference r : n) {
         System.out.println("  " + ref + " -> " + r);
49      }

        Message toSend = msg.createResponse();
        toSend.setTag(FeedbackTags.NEIGHBOURS_TAG);
        deliver(toSend, false, getManager());
54      // getManager() is equal to msg.getIssuer()
        // in this particular case
        return true;
       }
       case FeedbackTags.INIT_TAG: {
59      // Save message (to keep transactional information to
        // notify the end of the algorithm to the feedback manager)
        // Please refer to 'RETURN_TAG' case for 'initMsg' use
        initMsg = msg;
        value = (String) msg.getData();
64      System.out.println(getReference() + " - initiated \"" + value
         + "\"");
        Message toSend = new Message();
        toSend.setTag(FeedbackTags.MESSAGE_TAG);
        toSend.setData(value);
69      deliver(toSend, false, remainingNeighbours);
        return true;
       }
       case FeedbackTags.MESSAGE_TAG: {
        Message toSend = msg.createResponse();
74      synchronized (this) {
         remainingNeighbours.remove(msg.getIssuer());
         if (value != null) {
          toSend.setTag(FeedbackTags.RETURN_TAG);
          deliver(toSend, false, msg.getIssuer());
79        return true;
         }

         // Receiving value for the first time
         value = (String) msg.getData();
84      }

        System.out.println(getReference() + " - received \"" + value
         + "\"");
        parent = msg.getIssuer();
89      if (remainingNeighbours.isEmpty()) {
         toSend.setTag(FeedbackTags.RETURN_TAG);
```

```
        deliver(msg, false, parent);
        return true;
       }

       toSend.setTag(FeedbackTags.MESSAGE_TAG);
       toSend.setData(value);
       deliver(toSend, false, remainingNeighbours);
       return true;
      }
      case FeedbackTags.RETURN_TAG: {
       boolean empty;
       synchronized (this) {
        remainingNeighbours.remove(msg.getIssuer());
        empty = remainingNeighbours.isEmpty();
       }

       if (empty) {
        if (parent == null) {
         // Respond to feedback manager's initial message
         Message toSend = initMsg.createResponse();
         toSend.setTag(FeedbackTags.END_TAG);
         deliver(toSend, false, getManager());
        }
        else {
         Message toSend = msg.createResponse();
         toSend.setTag(FeedbackTags.RETURN_TAG);
         deliver(toSend, false, parent);
        }
       }

       return true;
      }
      default:
       return false;
     }
    }

    @Override
    public void terminateServer() {
    }
   }
```

Please note that distributed objects (graph nodes) are server-orientedas they must handle messages sent by other nodes.

7.1.4 Main program source code

```
1  /**
    * Copyright 2007 Nathanael Cottin
    */
4  package test;

   import java.util.Collection;
   import java.util.HashSet;

9  import sds.data.Message;
   import sds.data.Reference;
   import sds.error.SdsException;
   import sds.more.Miscellaneous;
   import sds.service.Daemon;
14 import sds.service.Manager;

   public final class FeedbackManager extends Manager {

    private final boolean[][] neighbours;
19  private final Reference  daemon;

    public FeedbackManager(boolean[][] neighbours, Reference daemon) {
     super(false);
     this.neighbours = neighbours;
24   this.daemon = daemon;
    }

    @Override
    public void execute() {
29   System.out.println("Executing FeedbackManager: " + getReference());

     System.out.println("Registering " + neighbours.length
      + " instance(s)");
     Collection<Reference> invalidRef;
34   try {
      invalidRef = register(FeedbackServer.class, neighbours.length,
       true, daemon);
     }
     catch (SdsException e) {
39    processError(e);
     return;
     }

     System.out.println("Invalid references: " + invalidRef.size());
44   for (Reference r : invalidRef) {
```

```
        System.out.println(r);
      }

      System.out.println("\n------------------------------------\n");
      Collection<Reference> ref =
        getDistributedObjectReferences(FeedbackServer.class);
      System.out.println("Registered references: " + ref.size());
      for (Reference r : ref) {
        System.out.println(r);
      }

      System.out.println("\n------------------------------------\n");
      Reference[] refs = Miscellaneous.toArray(ref);
      System.out.println("Sending neighbourhood to " + refs.length
        + " distributed object(s)");
      for (int i = 0; i < refs.length; i++) {
        sendNeighbours(refs[i], getNeighbours(refs, neighbours[i]));
      }

      System.out.println("\n------------------------------------\n");
      System.out.println("Initializing algorithm");
      Message msg = new Message();
      msg.setTag(FeedbackTags.INIT_TAG);
      msg.setData("Message to deliver");
      deliver(msg, true, refs[0]);

      System.out.println("\n------------------------------------\n");
      System.out.println("End: killing all distant processes");
      killAll(false);
    }

    @Override
    public void processError(SdsException error) {
      System.out.println("FeedbackManager encountered the following error:");
      error.printStackTrace();
    }

    @Override
    public boolean recv(Message msg) throws SdsException {
      switch (msg.getTag()) {
        case FeedbackTags.NEIGHBOURS_TAG:
          // Response to blocking neighbours delivery
          return true;
        case FeedbackTags.END_TAG: {
          // Response to blocking distributed algorithm execution
          return true;
```

```
      }
      default:
       return false;
    }
  }

  private void sendNeighbours(Reference recipient,
   Collection<Reference> n) {
   Message msg = new Message();
   msg.setTag(FeedbackTags.NEIGHBOURS_TAG);
   msg.setData(n);
   deliver(msg, true, recipient);
  }

  private Collection<Reference> getNeighbours(
   Reference[] refs, boolean[] n) {
   Collection<Reference> res = new HashSet<Reference>(n.length);
   for (int i = 0; i < n.length; i++) {
    if (n[i]) {
     res.add(refs[i]);
    }
   }

   return res;
  }

  @Override
  public void terminateServer() {
  }

  public static void main(String[] args) {
   try {
    // Graph incidence matrix with 3 nodes
    boolean[][] neighbours = { { false, true, false },
      { true, false, true }, { true, false, false } };

    // Launch daemon on localhost and port 666, and set distributed
    // objects ports from 1001 to 1004
    Daemon daemon = new Daemon(false, 1001, 1001 + neighbours.length,
     Daemon.DEFAULT_FIRST_ID);
    daemon.run(666, true);

    Reference daemonReference = daemon.getReference();
    FeedbackManager mgr = new FeedbackManager(neighbours,
     daemonReference);
    mgr.registerDaemons(daemonReference);
```

```
        mgr.run(1000);

139     // Also terminate manager and daemon to leave properly
        mgr.terminate();
        daemon.terminate();
       }
       catch (Exception e) {
144     e.printStackTrace();
       }
      }
     }
```

8 Known limitations

Please refer to *SDS* official website at `http://sds.ncottin.net`.