

Bionet Migration Service

Jun Suzuki, Kevin Nguyen and Tatsuya Suda

{jsuzuki, suda}@ics.uci.edu

nguyencp@uci.edu

<http://netresearch.ics.uci.edu/bionet/>

School of Information and Computer Science
University of California, Irvine

Bionet Migration Service

- Bionet migration service
 - is implemented within the package named:
 - `edu.uci.bionet.services.migration`
 - is responsible for sending out a CE and receiving an incoming CE.
 - runs on per-platform basis.
 - is used by only CEs running on a local platform.
 - No CEs cannot access a migration service that runs on a different (remote) platform.
- Each CE decide when and where to migrate by itself.
 - Any CE is not allowed to ask/force another CE to move.

2 Migration Mechanisms

- In general, there are 2 kinds of migration mechanisms.
 - (1) Strong migration
 - A CE migrates together with its whole execution state.
 - A CE's execution state contains all stack information that is required to characterize the point of execution that the CE has currently reached (e.g. local variable values within a method).
 - After a migration, the CE continues processing its task exactly at the point where it has been interrupted before the migration.
 - Need to modify Java VM to implement strong migration because it does not provides a way to capture the stack information within memory.

- (2) Weak migration
 - A CE migrates only with its data state when migrating.
 - A CE's data state contains values of instance variables (i.e. heap information).
 - No need to modify Java VM
 - Data state can be serialized at a source host (before the migration), transferred across network, and deserialized at a destination host (after the migration).
- The bionet migration service supports weak migration because it is simpler and more portable.
- Based on weak migration, a migration involves
 - serializing (creating data state of) a CE,
 - sending out the CE's data state,
 - destroying the CE at a source platform, and
 - creating a new CE (based on the transferred state) at a destination platform.

Information Transferred in Migration

- The bionet migration service transfers
 - CE's (data) state
 - values of instance variables that are not declared as transient.
 - obtained through Java serialization mechanism.
 - CE's class definition (the content of a class file)
 - obtained through `java.lang.Class`.
 - CE's class name
 - obtained through `java.lang.Class`.

Interface/Class Definition

- The bionet migration service consists of 2 components; *sender* and *receiver*.
 - Sender is implemented with Java interface and class.
 - `interface CESender (Java)`
 - `class CESenderImpl (Java)`
 - Receiver is implemented with CORBA IDL interface and Java class.
 - A receiver receives Java mobile code through CORBA.
 - `interface CEReceiver (CORBA)`
 - `class CEReceiverImpl (Java)`

- **Sender**

```

package edu.uci.bionet.services.migration;
// Java interface
public interface CESender{
    void moveTo( MigrationServiceRef remoteMigrationService );
}
// Java class implementing the above Java interface
public class CESenderImpl implements CESender{
    void moveTo( MigrationServiceRef remoteMigrationService );
}

```

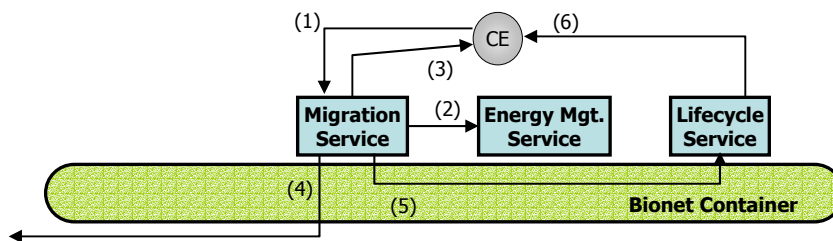
- **Receiver**

```

// CORBA IDL
typedef sequence<octet> classdef;
typedef sequence<octet> datastate;
interface CEReceiver{
    void receive( in classdef class, in datastate state, in string
name);
}
// Java class implementing the above Java interface
public class CEReceiverImpl{
    void receive ( byte[] class, byte[] state, String name );
}

```

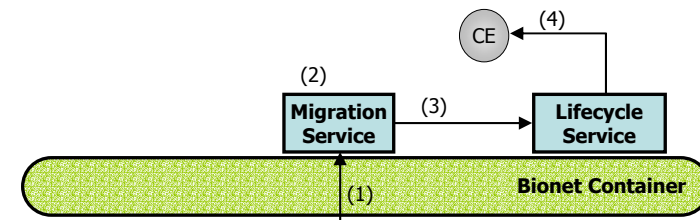
Step-by-step Behavior of Bionet Migration Service



- **Procedure to send out a CE**

- (1) A CE calls `moveTo(MigrationService remoteMigrationService, CEImpl currentCE)` of a local bionet migration service.
 - It specifies a reference of a bionet migration service running on a destination platform.
- (2) The bionet migration service checks if the CE has enough energy to migrate by referring to an energy table in a bionet energy management service on the source platform.

- (3) The bionet migration service calls `onMove()` on the CE.
 - It is left to CE developers how to implement `onMove()`.
 - An example is to emit the CE's pheromone on the local platform.
- (4) The bionet migration service
 - serializes the CE's data state, which includes values of instance variables using Java serialization mechanism,
 - specifically, using `java.io.ObjectOutputStream` and `java.io.ByteArrayOutputStream` to convert data state into byte sequence
 - transforms the CE's class definition (i.e. class file) into a sequence of byte stream (i.e. the data typed as `byte[]`), and
 - using `java.io.ObjectOutputStream` and `java.io.ByteArrayOutputStream`
 - transfers the CE's serialized data state, class definition and class name to a destination bionet migration service.
 - through `receiveCE(String ceName, byte [] classDef, byte [] dataState)` of the destination migration service
- (5) Bionet migration service calls `destroy()` of a bionet lifecycle service (on source platform) to remove the CE from the source platform.
- (6) Bionet lifecycle service calls `onDestroy()` on the CE.
 - It is left to CE developers how to implement `onDestroy()`.



- Procedure to receive an incoming CE
 - (1) A bionet migration service at a destination platform receives a mobile code (i.e. a migrating CE's data state, class definition and class name) through its operation `receiveCE(String ceName, byte [] classDef, byte [] dataState)`.

- (2) The bionet migration service
 - receives a mobile mode through its operation, receiveCE(String ceName, byte [] classDef, byte [] dataState),
 - loads the received byte sequence of class definition into a local JVM using our own custom class loader
 - edu.uci.ics.bionet.util.classloader.MigrationClassLoader
 - See the class loader documentation for more details about class loading
 - deserializes the received data state into an instance by using Java serialization mechanism , and
 - specifically, using java.io.ObjectInputStream and java.io.ByteArrayInputStream to convert the received byte sequence of data state into a java object
 - downcasts the created (deserialized) instance from `java.lang.Object` into `CyberEntityImpl`, which is the super class for all the CEs.
- (3) The bionet migration service calls `initialize()` of local bionet lifecycle service (running on a destination platform).
- (4) The bionet lifecycle service calls `onArrival()` on the CE.
 - It is left to CE developers how to implement `onArrival()`.