

Bionet Lifecycle Service

Bionet Lifecycle Service

Jun Suzuki, Michael Le and Tatsuya Suda

{jsuzuki, suda}@ics.uci.edu

mvle@uci.edu

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

Dept. of Information and Computer Science
University of California, Irvine

- Bionet lifecycle service is implemented within the package named:
 - `edu.uci.ics.bionet.services.lifecycle`
- This service takes care of lifecycle of the CEs that exist on a local platform.
 - The service does not control the lifecycle of CEs that run on a remote platform.
- A bionet lifecycle service is implemented as a set of Java interfaces and classes (not CORBA objects).
 - runs on per-platform basis.
 - accessed by only CEs running on a local platform.
 - CEs are not allowed to access bionet lifecycle services that run on different (remote) platforms.

Functionalities of Bionet Lifecycle Service

Bionet lifecycle service

- allows a CE to change its state.
- maintains a thread pool that contains a certain number of threads that can be assigned to autonomous CEs.
- allows a CE to replicate itself.
 - Mutation can happen.
- allows a CE to reproduce a child CE with a partner.
 - Mutation and crossover can happen.
- upcalls callback methods on a CE.

Interface and Its Implementation of Bionet Lifecycle Service

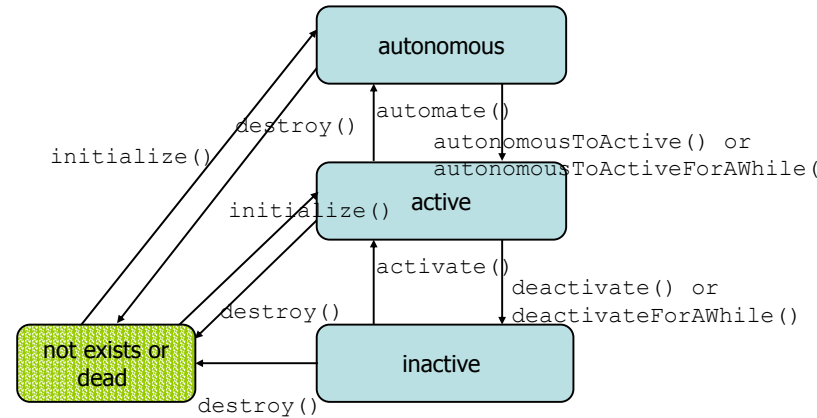
```
• package edu.uci.ics.bionet.services.lifecycle;
  interface LifecycleService
  {
      public org.omg.CORBA.Object initialize(
          CyberEntityImpl aCE,
          double initialEnergyAmount,
          String context,
          String geneInfo,
          String initialState)
          throws NoIdleThreadException, WaitingRunnableQueueBlocked, CEInitErrorException;
      public void destroy(org.omg.CORBA.Object ref);
      public void automate(org.omg.CORBA.Object ref)
          throws NoIdleThreadException, WaitingRunnableQueueBlocked;
      public void autonomousToActive(org.omg.CORBA.Object ref);
      public org.omg.PortableServer.Servant activate(byte [] oid);
      public void deactivate(org.omg.CORBA.Object ref);
      public void autonomousToActiveForAWhile(org.omg.CORBA.Object ref,int sec);
      public void deactivateForAWhile(org.omg.CORBA.Object ref, int sec);
      public org.omg.CORBA.Object replicate(
          org.omg.CORBA.Object parentRef,
          double initialEnergyAmount,
          String initialState);
      public org.omg.CORBA.Object reproduce(
          org.omg.CORBA.Object parentRef1,
          org.omg.CORBA.Object parentRef2,
          double initialEnergyAmount,
          String initialState);
      public void shutdown();
  }
```

CE's Internal state

```
package edu.uci.ics.bionet.services.lifecycle;  
class LifeCycleServiceImpl implements LifeCycleService  
{  
    // implementation methods of the above 10 interface operations  
    // defined in LifeCycleService.  
}
```

5

- The operations defined in bionet lifecycle service trigger the transitions of CE's state.
 - 8 interface operations
 - 4 states



LifeCycleServiceImpl Constructor

- **public LifeCycleServiceImpl ();**
 - This operations is called when a LifeCycleService is first constructed
- Action sequence in this operation
 - obtain the directory path to store and retrieve serialized CEs from the Java system property.
 - The key is “serializedce.dir”.
 - c.f. documentation about platform configuration properties
 - obtain the maximum number of threads in the threadpool from the Java system property.
 - The key is “maxthread.num”.

Each CE changes its state voluntarily; any CEs are not allowed to change another CE's state.

An autonomous CE

- receives messages from other cyber-entities, and autonomously processes them.
- continuously senses nearby environment and performs its behaviors accordingly.
- runs on an individual thread; thus, expends energy continuously for using a thread (CPU) and memory space.

An active CE

- receives messages from other cyber-entities, but
- does not process the received messages.
 - i.e. Only autonomous CEs process the received messages.
- does not continuously sense nearby environment and perform its behaviors
- consumes memory; thus expends energy continuously for using memory space.

An inactive CE

- is in “sleeping” state in which it is externalized into a file

initialize()

- create the singleton threadpool object with the obtained maximum thread number.
- obtain a reference to the local platform representative.
 - The platform representative is the singleton class; its singleton instance can be obtained through its static getInstance() method.
- obtain a reference to the bionet container through the local platform representative.

9

- `org.omg.CORBA.Object initialize(CyberEntityImpl aCE, double initialEnergyAmount, String context, String geneInfo, String initialState);`
- This operation is used to make a CE workable and available on the bionet environment.
 - An instantiated CE is specified as a parameter of initialize().
 - A CORBA object reference to the initialized CE is returned from initialize().
- This operation is called by
 - a CE developer (when a CE is manually created),
 - a bionet migration service (when a CE moved from another platform or
 - replicate() or reproduce() operation of a bionet lifecycle service (when a CE replicated/reproduced a child CE).

Action sequence in initialize()

- makes sure there is an idle thread before doing any work
 - c.f. the documentation about thread pool
- creates a CE context, and associates it with the CE by calling the CE's setCEContext()
 - c.f. the documentation about CE context
- acquires genetic information from a formatted file for newly created CE
 - c.f. the documentation about gene-injected CE
- creates the CE's GUID
 - c.f. the documentation about GUID generator
- registers the CE into a CE table of a bionet container
 - bionet container = POA
 - A CE table = active object map
 - i.e. calls POA::activate_object_with_id()

- creates the CE's reference using a bionet container
 - i.e. calls POA::create_reference_with_id()
 - CE's GUID is used as a parameter of this operation (i.e. object id).
- assigns an initial amount of energy (specified as a parameter of initialize()) to the CE
 - This amount may be specified by a human developer or parent CE(s)
 - creates an entry of an energy table in a bionet energy management service.
 - c.f. bionet energy management service
- If the `context` parameter of this operation is "MIGRATION"
 - This means that the CE was not created from scratch but migrated from another platform.
 - calls `onArrival()` on the CE
 - Implementation of onArrival() is left to CE developers.

- If the `context` parameter of this operation is `"REPLICATION"`,
 - This means that the CE was replicated from a parent CE.
 - calls `onReplicated()` on the CE.
- If the `context` parameter of this operation is `"REPRODUCTION"`,
 - This means that the CE was reproduced from parent CEs.
 - calls `onReproduced()` on the CE.
- If the `context` parameter of this operation is `"FROM_SCRATCH"`.
 - This means the CE was created from scratch.
 - calls `onCreated()` on the CE.
 - Implementation of `onCreated()` is left to CE developers. Its typical implementation is to create the CE's metadata and register the CE to a local directory.

13

- If `context` parameter of this operation is `"AUTONOMOUS"`,
 - The CE is initialized as an autonomous one.
 - calls `automated()`
 - Check if thread pool have any idle thread.
 - If so then:
 - » call `onAutomated()` which will set `CEState` to autonomous
 - » assigns an idle thread in a thread pool to the CE
 - » The CE's `run()` method will be invoked.
 - Else:
 - » Throw an exception when there is no idle threads
 - » Throw `NoldleThreadException`
- If `context` parameter of this operation is `"ACTIVE"`,
 - The CE is initialized as an active one.
 - calls `activated()`

automate()

void automate(Object ref);

- This operation is used to change the state of the specified CE from active to autonomous.
- A CE calls this operation voluntarily; any CE is not allowed to call it to change another CE's state.
- The specified parameter is a CORBA object reference.

Action sequence in this operation

- calls `reference_to_servant()` of bionet container (POA) to obtain a pointer (servant) to the specified CE (CORBA object)
- Typecast servant into `CyberEntityImpl` in order to pass it into the thread pool, which accepts object of type `Runnable`.

- Check if thread pool have any idle thread.
 - If so then:
 - call `onAutomated()` which will set `CEState` to autonomous
 - assigns an idle thread in a thread pool to the CE
 - The CE's `run()` method will be invoked.
 - Else:
 - Throw an exception when there is no idle threads
 - » Throw `NoldleThreadException`

autonomousToActive()

```
void autonomousToActive(Object ref);
```

- This operation is used to change the state of the specified CE from autonomous to active.
- The specified parameter is a CORBA object reference to a CE.
- The CE that changes its state from autonomous to active will become autonomous again, if it calls `automate()`.
- If it wants to be autonomous again in a certain time period, it should call `autonomousToActiveForAWhile()`, instead of `autonomousToActive()`.

Action sequence in this operation

- stops executing the specified CE's `run()` method
- call `onAutonomousToActive()`, which will break a loop that is executed in `run()` by setting the `CEState` to active

17

autonomousToActiveForAWhile()

```
void autonomousToActiveForAWhile(Object ref,int sec);
```

- This operation is used to change the state of the specified CE from autonomous to active for the specified time period.
- A CE calls this method voluntarily; any CE is not allowed to make another CE become active.
- The specified parameter is a CORBA object reference.
- If it wants to be active permanently, it should call `autonomousToActive()`, instead of `autonomousToActiveForAWhile()`.
- Action sequence in this operation
 - calls `autonomousToActive()`
 - creates a timer thread which counts the specified time period
 - When the timer thread finishes counting the specified time period, it will invoke `automate()` method to activate the CE again.

activate()

```
org.omg.PortableServer.Servant
```

```
activate(ObjectId oid);
```

- This operation is used to change the state of the specified CE from inactive to active.
- This operation is called by `incarnate()` of `CEActivator`.
 - When a message arrives to an inactive CE, a bionet container calls `incarnate()` on its `CEActivator`, which in turn calls `activate()`.
 - `activate()` is called only by bionet container.
 - Any CEs do not call this operation
 - » In the current design, a CE is activated (by bionet container) only when it receives a message from another CE.
 - `CEActivator` is a servant activator on a POA (a bionet container).

Action sequence in this operation

- checks if the specified CE has enough energy to activate (by using a bionet energy management service).

- If the specified CE has enough energy...

- asks a bionet energy management service to update the CE's entry in an energy table (changes its flag from inactive to active).
- de-serialize the CE from a file (using Java serialization mechanism), and cast the de-serialized object into `CyberEntityImpl` type.
- the file name is the same as the CE's object id
- the full path of the serialized CE file is obtained by concatenating the serialized CE directory and the CE's object id (filename)
- call `onActivated()` on the CE.
 - Implementation of `onActivated()` is left to CE developers.
 - At least should set the data field `CEState` to active
- return the `CyberEntityImpl`

deactivate()

```
void deactivate(Object ref);
```

- This operation is used to change the state of the specified CE from active to inactive.
- CEs call this operation voluntarily; CEs are not allowed to deactivate another CE.
 - Each CE decides when to deactivate.
- A deactivated CE will be activated when it receives a message from another CE.
 - If it does not receive a message, it will not be activated.
 - If a CE wants to be activated again in a certain time period, it should call `deactivateForAWhile()`, which is described in the next slide.

Action sequence in this operation

- asks a bionet energy management service to update the CE's entry in an energy table (i.e. changes its flag from active to inactive).
- calls `reference_to_servant()` of bionet container (POA) to obtain a pointer to the specified CE (i.e. servant)

21

- calls `reference_to_id()` of bionet container (POA)
- serializes the CE into a file and use the object ID as filename
 - All the serialized CEs are put in the serialized CE directory obtained through the Java's system property.
- calls `onDeactivated()`
 - Implementation of `onDeactivated()` is left to CE developers.
 - Should at least set the data field `CEState` to inactive
- calls `deactivate_object(ObjectId oid)` of bionet container (POA)
 - When `deactivate_object()` is called, a bionet container (POA) invokes `etherealize()` on a `CEActivator` (servant activator) to remove the CE from the bionet container.

deactivateForAWhile()

```
void deactivateForAWhile(  
    Object ref,  
    int sec);
```

- A CE calls this method, if it wants to be inactive in a certain time period and then become active again.
 - CEs call this method voluntarily; any CEs are not allowed to deactivate another CE.
 - Each CE decides when to deactivate.

Action sequence in this operation:

- calls `deactivate()`
- creates a timer thread which counts the specified time period
 - When the timer thread finishes counting the specified time period, it will invoke `activate()` method for activating the CE again.

destroy()

- **void destroy(Object ref);**
 - This operation is used to delete the specified CE.
 - CEs can call this operation voluntarily; Any CEs are not allowed to destroy other CEs.
 - A bionet energy management service can also call this operation, when energy level of a CE becomes zero.
- **Action sequence in this operation**
 - returns the CE's thread to the threadpool, in case it is at autonomous.
 - calls `onDestroyed()` of the CE
 - Implementation of `onDestroyed()` is left to CE developers.
 - At least should set the data field `CEState` to destroyed and break `run()` and `break run()` loop for returning a thread to the thread pool

- deletes an entry regarding the specified CE from a bionet energy management service
- Remove ObjectID and pointer of the CE from Active Object Map by calling deactivate_object(), which in turn will call etherealize()

25

replicate()

- `org.omg.CORBA.Object replicate (Object parentRef, double initialEnergyAmount, String initialState) ;`
 - This operation is used to replicate the specified CE.
 - CEs can call this operation voluntarily; Any CEs are not allowed to replicate other CEs.
- Action sequence of this operation
 - checks if the specified (parent) CE has enough energy to replicate (by using a bionet energy management service).
 - makes a copy (i.e. clone) of the specified (i.e. parent) CE, if it has enough energy
 - makes a deep copy of the parent CE (not shallow copy) using the Java serialization mechanism.

- calls execMutation() on the child CE to execute mutation.
 - c.f. the documentation about gene-injected CEs
- casts replicated object into CyberEntityImpl
- calls initialize()
 - “REPLICATION” is assigned to the third parameter of this operation.
 - “AUTONOMOUS” is assigned to the fourth parameter.
 - initialize() will, in turn, invoke onReplicated() on the replicated CE.
 - Implementation of onReplicated() is left to CE developers.
 - » e.g. Some relationships that the parent CE has may be inherited, and some may not.
- returns the child CE’s reference, that was returned

reproduce()

- `org.omg.CORBA.Object reproduce (Object parent1Ref, Object parent2Ref, double initialEnergyAmount, String initialState) ;`
 - This operation is used to reproduce a child CE from the specified two parent CEs.
 - CEs can call this operation voluntarily; Any CEs are not allowed to destroy other CEs.
- Action sequence in this operation
 - checks if the specified CEs (parents) have enough energy to reproduce (by using bionet energy management service).
 - makes a copy (child CE) of the parent1 CE, if it has enough energy
 - makes a deep copy of the parent CE (not shallow copy) using Java serialization mechanism.

shutdown()

- casts the child CE to CyberEntityImpl type
- calls execCrossover() on the child CE to execute crossover.
- calls initialize()
 - “REPRODUCTION” should be the third parameter of this operation.
 - “AUTONOMOUS” should be the fourth parameter.
 - initialize() will invoke onReproduced() on the reproduced CE.
 - Implementation of onReproduced() is left to CE developers.
 - » e.g. Some relationships that the parent CE has may be inherited, and some may not.
 - return the child CE’s reference, that was returned from having called initialize()

29

- Ultimately shutting down the Bionet platform.
 - shut down the threadpool
 - completely remove all running CEs
 - disallow any new addition of CEs into the threadpool.