CyberOrgs 1.0 Manual

Xinghui Zhao
Agents Lab
Department of Computer Science
University of Saskatchewan
xinghui.zhao@agents.usask.ca

1 Overview

CyberOrgs [1] is a model for hierarchical coordination of resource usage by multi-agent applications in a network of peer-owned resources. Each cyberorg encapsulates a set of computations which are executed concurrently, and an amount of resource. A concurrent computation consumes resource, which is allocated to it by its containing cyberorg. A cyberorg has a contractual relationship with its containing cyberorg, and it may purchase resources from its containing cyberorg according to the signed *contract*. The currency that flows among cyberorgs is called eCash.

CyberOrgs organizes resources and computations as a tree. Each cyberorg except the root cyberorg is contained inside another cyberorg. Figure 1 shows the hierarchical structure of a cyberorg. Black dots represent computations, and ellipses represent cyberorgs. A cyberorg hosted by another cyberorg purchases resources it needs from the host cyberorg, according to a pre-negotiated contract. This contract, which must be signed between two cyberorgs before one is hosted by the other, stipulates the types and quantities of resources which will be available to the hosted cyberorg as well as their costs. After satisfying its contractual obligations, a cyberorg distributes the remaining resources available to it among the computations it is managing according to its own local resource distribution strategy.

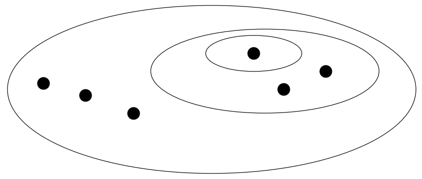

Figure 1: Hierarchical Structure of a Cyberorg

CyberOrgs distribute resources through several primitives: *isolate*, in which a new cyberorg is created, and its host cyberorg delegates resource control to it; *assimilate*, in which a cyberorg relinquishes its resource control to its host cyberorg; and *migrate*, which is used by a cyberorg to pursue a better host cyberorg.

CyberOrgs1.0 is a Java implementation of CyberOrgs model, and it is designed to provide fine-grained resource control for distributed applications. CyberOrgs1.0 is built on top of *Actor Architecture* [2], which is a Java library and run-time system for supporting actor execution and primitives.

Figure 2 shows the design of the CyberOrgs implementation. We extend Actor Architecture by adding two key components: *CyberOrg Manager* and *Scheduler Manager*. CyberOrg Manager is the central component of each CyberOrgs platform. All resource control operations are carried out by CyberOrg Manager. The results of such operations are sent to Scheduler Manager, which schedules all actors in the platform according to these results.

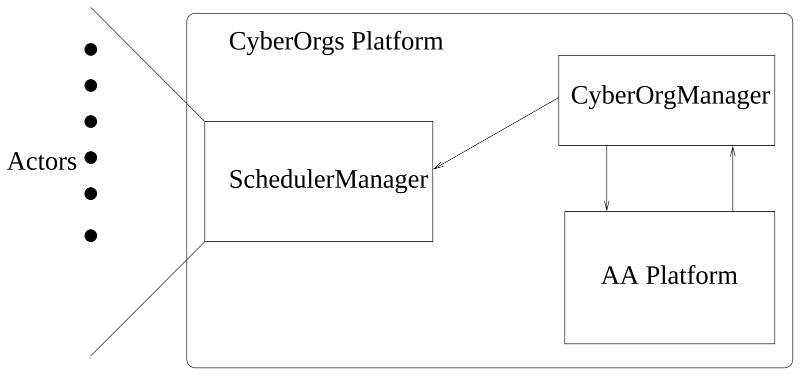

Figure 2: CyberOrgs System Design

2 Installation Guide

CyberOrgs1.0 is a Java implementation, and it is platform independent. To install the software, please follow the instructions:

- Download the source package from http://agents.usask.ca
- Extract the zip file in the current folder
- For Linux or Mac OS, use the make file to build the system; for other operating systems, use the following commands to recompile:

  ```
  ...>CyberOrgs>javac */*.java
  ...>CyberOrgs>javac */*/*.java
  ...>CyberOrgs>javac */*/*/*.java
  ```

- The CyberOrgs system should now be ready to use

3 Getting Started

There are a few trivial examples in the package illustrating how to use the software. We introduce two examples in this section.

3.1 Fibonacci

In this example, we illustrate how to implement a simple actor application using CyberOrgs system. The default resource allocation policy is applied in this application.

The application is aimed to calculate *Fibonacci numbers*, which are defined as follows:

$$F(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F(n-1) + F(n-2) & \text{if } n > 1 \end{cases}$$

The implementation is located in the package `aa.application.Fibonacci`. There are two classes, `FibActor` and `BootFib`. The former defines the behavior of a Fibonacci worker actor, and the latter is the main program of the application. A `FibActor` receives a number from user's input or its creator, calculates Fibonacci function by creating two more actors and sending initial numbers to them, and waits for the results from the two new actors.

The Fibonacci example does not define a specific resource allocation policy, therefore the default policy is applied. Under the default resource allocation policy, every newly created actor will receive a random amount of CPU resources from the cyberorg.

Please note in this example, no cyberorg primitive operation is triggered. The whole computation takes place in the root cyberorg. The main purpose of this example is to illustrate how to implement a simple application which uses the default resource allocation policy of the CyberOrgs system.

3.2 Cyberorg mobility

This example shows cyberorg's mobility. Two physical nodes are involved in this application, *source* node and *destination* node. In this example, we create a new cyberorg inside the root cyberorg using *isolate* primitive, then we trigger isolate primitive again in the newly created cyberorg, which becomes a nested cyberorg, as shown in Figure 3. The nested cyberorg will be migrated from the source node to the destination node using *migrate* primitive.

The implementation of cyberorg mobility is in the package `aa.dirvers`. The class `FaciPara` is the definition of the facilitator actor in the mobile cyberorg; `driverDesti` is the main program running on the destination node; and `driver` is the main program on the source node.

The instruction of executing the example is as follows. First, start the main program `driverDesti` on the destination node, where a CyberOrgs platform will be initialized. Second, start the main program `driver` on the source node.

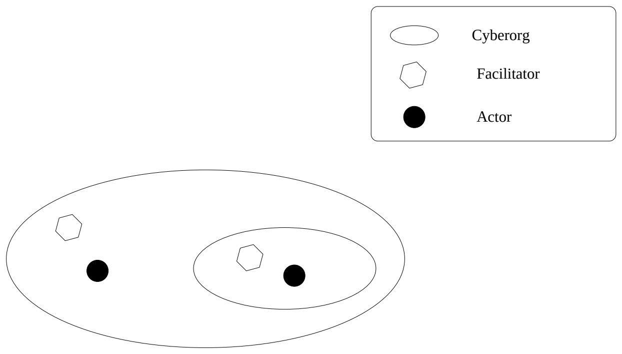

Figure 3: The Structure of the Migrating Cyberorg

It prompts the user to input the IP address of the destination node, then the migration process will be started.

4 Programmer Guide

Programmers can develop their own resource allocation policy for the cyberorgs in their applications. *Adaptive Quadrature* is an example of user-defined resource allocation policy.

4.1 Adaptive quadrature

Adaptive quadrature is a classic problem in mathematics. In this problem, we seek to compute the integral of a given real-valued function $f(x)$ on a finite interval $a \leq x \leq b$:

$$\int_a^b f(x)dx$$

The implementation can be found in the package `aa.application.AdaptiveQuad`. Application actors are implemented, and they are responsible for calculating the quadrature value of a given function $f(x) = x \times sin(1/x)$ according to the fundamental additive property. We implemented several different resource allocation mechanisms in this package, and for each individual mechanism, we defined a specific application actor. Although the application actors have the same computational function, they are slightly different because of the resource allocation plan.

`QuadActor_noRes` and `bootQuad` implement an adaptive quadrature computation which takes the CyberOrgs default resource allocation policy; `QuadActor_Res`, `QuadFacilitator` and `BootQuad_Res` represent the same adaptive quadrature computation, however, it uses a user defined resource allocation policy, which will be described in Section 4.2; `QuadActor_RemoteRes`, `QuadFacilitator`, `RemoteRes_DestinationNode`, and `BootQuad_RemoteRes` define a multi-node adaptive quadrature application, which takes advantage of CyberOrgs mobility. The detail description can be found in Section 4.3

4.2 Resource allocation policy

We define a new resource allocation policy for controlling resources in the adaptive quadrature application. When a new application actor is created inside one cyberorg, this cyberorg may isolate it to a new client cyberorg if the new actor's error is greater than a threshold.

We extend the `Facilitator` class to control resource, and we use the `isolate` primitive to encapsulate the new actor (which has relatively large workload) to a new cyberorg in order to guarantee resource supply. In particular, if the ratio of the new actor's error to the average error so far exceeds some threshold, we set aside some eCash and isolate this new actor to a new cyberorg; and if not, we use the resource allocation policy of the original cyberorg to assign resources to the new actor.

Figure 4 shows the `resourceAllocate` method in the `Facilitator` class.

```
public void resourceAllocate(ActorName newActor, double error){
  ratio = error/averageError;
     //compute the ratio of error to the average error value of all actors
  if (ratio > threshold){
    myeCash = averageeCash * ratio;
    /* generate a contract for isolation */
    hostCyberOrg.isolate(eCash,newActor,myContract);
       //tell the host cyberorg to isolate the new actor to a child cyberorg
    /* update the value of averageError */
    /* update the value of averageeCash */
  }
 else{
    hostCyberOrg.resourceAllocate(newActor);
     //tell the host cyberorg to carry out the resource allocation
    /* update the value of averageError */
    /* update the value of averageResource */
 }
}
```

Figure 4: Method `resourceAllocate` in Facilitator Actor of the cyberorg in the Adaptive Quadrature Example

4.3 Load balancing using CyberOrg mobility

In this implementation, two physical nodes are involved. On each CyberOrgs platform, we use the same resource allocation policy as described in Section 4.2. However, if the error of a new application actor is greater than a threshold, this part of computation represented by the application actor will be moved to the other platform, in order to balance the workload between the two nodes.

The instruction of executing the example is as follows: first, start the main program `RemoteRes_DestinationNode` on the backup node; second, start the main program `BootQuad_RemoteRes` on the main node, and input the IP address of the backup node when prompted to do so. Then the computation will begin, and it will be automatically divided into two parts running on two nodes.

5 Limitations and Ongoing Work

CyberOrgs1.0 does not support network resource control. Work is ongoing to investigate the multiple type resource coordination over a large scale distributed system.

CyberOrgs is a research project in Agents Lab at University of Saskatchewan, and it is still under development. If you have any questions or comments, please report to xinghui.zhao@agents.usask.ca.

References

[1] Nadeem Jamali. *CYBERORGS: A Model for Resource Bounded Complex Agents*. PhD thesis, University of Illinois at Urbana-Champaign, 2004.

[2] Open System Laboratory. The actor architecture. *Available for download at(http://osl.cs.uiuc.edu/aa)*, 2004.