Architecture of the Object Oriented Anonymous Remote Computing Framework for C# over .NET[*]

T. Vamsi Kalyan, R.K. Joshi
Department of Computer Science and Engineering
Indian Institute of Technology, Bombay

Abstract

Anonymous Remote Computing (ARC) for C# over .NET is a service oriented framework to support development of parallel and distributed programs in presence of mobility. The paper provides an introduction to the framework and discusses the architecture from use cases, services and components point of views. The application development process is also discussed. The basic design features of this framework are services for parallelism and distribution at object level. These include distribution of ARC objects, support for dynamic load measurement, fault tolerance and dynamic leave and join services for participating machines. In addition to these basic services, the framework is extended to support multiple object hoppings and object accessibility in presence of mobility. The design and implementation of ARC kernel are both based on object-oriented technology.

1 Introduction

Anonymous Remote Computing(ARC) [4] is a framework to support development of parallel programs over a cluster of workstations in presence of heterogeneity, load and failures. In an ARC environment, participating nodes may join and leave the system dynamically. In [1], a design and implementation of a service-oriented ARC kernel over a LINUX cluster has been discussed. Services are modeled through RPC based protocols. Though the modeling of this kernel was done through object modeling techniques, the implementation is procedural. Interfaces identified during modeling are translated into C-based RPC services during implementation.

In this paper, an ARC framework for C# over .NET is presented. In addition to the above services, support for object level mobility and retraction services is provided. The framework is modeled using object oriented analysis and design techniques, and also uses object oriented implementations. Object orientation at implementation level provides interesting solutions such as automatic-proxy switching for object mobility, implementation of multiple interfaces for services and factories for object creation.

Mobile objects introduced in the C# ARC framework are self-contained autonomous objects which can move from machine to machine, performing assigned task. While object move and perform tasks on remote machines, they remain addressible for the originator through connection, and also to the current context where the object moves. A migrated object may decide to hop to another

[*]This work is sponsored by Microsoft Inc. under an R&D grant at Department of Computer Science and Engineering, IIT Bombay.

machine or the originator application may retract the object, or the object may be pushed onto another machine in its next hop by the originator or by its current context.

2 Framework Capabilities

Various features of the ARC framework are discussed in this section. First the features are introduced and then the architecture is discussed in next section. Figure 1 shows a use-case diagram [2] bringing out the functional view of the ARC framework. Two kinds of actors namely *Parallel/Distributed Application* and *Node Administrator* can be noted. Parallel or distributed applications use constructs provided by ARC system to develop programs involving more than one machine. Node administrator deals with join and leave services. The functionalities identified in the use case diagram are elaborated below.

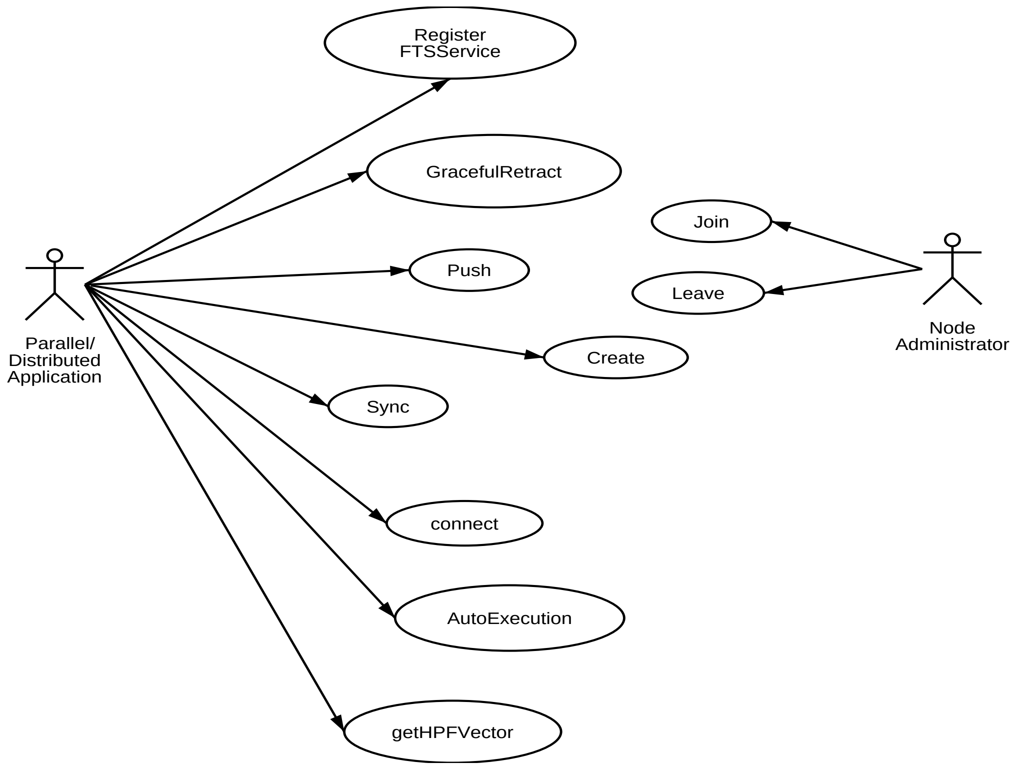

Figure 1: Use-Case Diagram for ARC

- Specifying an ARC object: An object is specified as an ARC object through inheritance. An ARC object obtains all the capabilities provided by the ARC system.

- Anonymity in Node selection: An application may select a machine by its *Horse Power Factor (HPF)* value, while the machine remains anonymous for the application. HPFs may be obtained via a call to *getHPFVector()* to a local HPF server. An HPF value is an instance of a class. An HPF server communicates with other HPF servers in the network.

- Explicit node selection: An application may also select a machine explicitly for migration through a call to *getHPFValue() on local HPF server.*

- Migration Assurance: When an application obtains an HPF value of a machine, or a vector corresponding to a set of machines, each value represents an assured slot for receiving a migrating object. The slots are unlocked through a call to *release()* on the local HPF server.

- Migration: Migration of an ARC object to an anonymous machine. An ARC object may be migrated by calling a method *push()* on the object. The method requires an HPF value corresponding to an anonymous machine as its parameter.

- Trigger: A method body to be executed after the object moves to remote machine is implemented as an overridden implementation of method *trigger()* by an arc-object.

- Parallelism: While an application migrates an ARC-object to an anonymous remote machine due to a call to *push*, it may continue in a non-blocking fashion. A result of remote execution available as migrated object's state.

- Auto-retraction: Asynchronous return of an object after completing the task at remote machine is a default retraction mechanism. After the object retracts, all invocations on the object are performed locally. The application remains unaware of the location of the object due to an internal automatic proxy switching mechanism.

- Wait till retraction: An application may wait on an ARC object till it is retracted through a call to *sync()* on the ARC object. If the object is already retracted, the call is unblocked immediately.

- Explicit Retraction: An object may be called back to originator when required through a call to *GracefulRetract()* on the ARC object. The remote ARC object is intimated via a flag which may be checked through a method *isRetractionSet()*. The trigger method in an ARC object may be implemented to handle an incoming graceful retract request.

- Roaming: An object may re-hop over a network of machines by means of a call to *push()* on the object. This call can be made by the originator, by its current context or by the object itself. The protocol followed by *push* is the same as that followed on its originating machine. For the originating application code, the object's location need not be known for call invocations on the roaming object. To establish this communication, the originating application needs to call a *connect()* invocation on its local ARC object handle. Proxy switching is performed internally in response to connect.

- Fault Tolerance: A desired fault tolerant behavior for an ARC object may be specified. Before an ARC object is migrated, to access a failure recovery service, the object is registered with a fault tolerance service (FTS) through a call to *register()* specifying the desired fault tolerance semantics for a given ARC object. An FTS server communicates with remote FTS servers for failure detection. Upon a failure detection, the FTS carries out resend operations through local ARC system interface.

- Communicating Mobile Objects: Roaming objects may exchange messages between each other. If the location of an object is known, an explicit proxy to the object may be obtained. For communication among roaming objects their locations have to be known to each other. An explicit proxy is required for all contexts other than the originating code for communication with a roaming object.

- Dynamic join and leave: Machines may join or leave an ARC system dynamically through a Join service and a Leave program. Every active participant exports a copy of the Join server. The list of currently active nodes is updated by join service on every machine.

3 Architecture

Figure 2 depicts the architecture of ARC framework. The ARC system is organized in the four layers which are described below.

3.1 ARC User Upper Layer

This layer consists of user programs running on the ARC system. These programs are distributed and parallel applications or node administrators. Distributed and parallel applications use services meant for user applications. Node administrators are responsible for joining a machine into the ARC network as well as disconnecting the machine from ARC network.

3.2 ARC User Lower Layer

Classes in this level include proxies for the ARC services and classes which may be generated through a preprocessor operating on interface descriptions. The inheritance structure of these classes is shown subsequently. Class *RealObj* is at the lowest level of inheritance and user implements the interface in this class. This layer also consists of proxies to services provided in ARC kernel level. These are obtained using .NET remoting infrastructure.

3.3 ARC Kernel Layer

This layer provides services to the layers above it. Depending on the user of the services, these services are classified into 3 sub parts.

- **ARC Object Services:** It allows registration of newly created ARC objects, Migration (send and receive) of code and state, and activation and execution of trigger on a newly arrived remote ARC object. When a migrated ARC object returns asynchronously, the object is inserted into its original location by proxy switching through ARC object proxy layer. ARC object proxy layer consists of proxies to locally registered ARC objects. The proxy layer is essential since the ARC object services are located in a different address space than that of the user programs.

- **Developer Services:** Distributed or parallel programs uses these services through proxies, which can be obtained using .NET remoting framework. These services can also be used from ARC user lower layer. Services supported in this category are fault tolerance and auto execution, Horse Power Factor and remote slot locking and unlocking, and asynchronous object arrival intimation services.

- **Node Administrator Services:** Node administrator can join a machine into the ARC network and also can unsubscribe a machine from the ARC network. Node administrator services include join and leave services. An active ARC computation uses actively participating nodes at a given time.

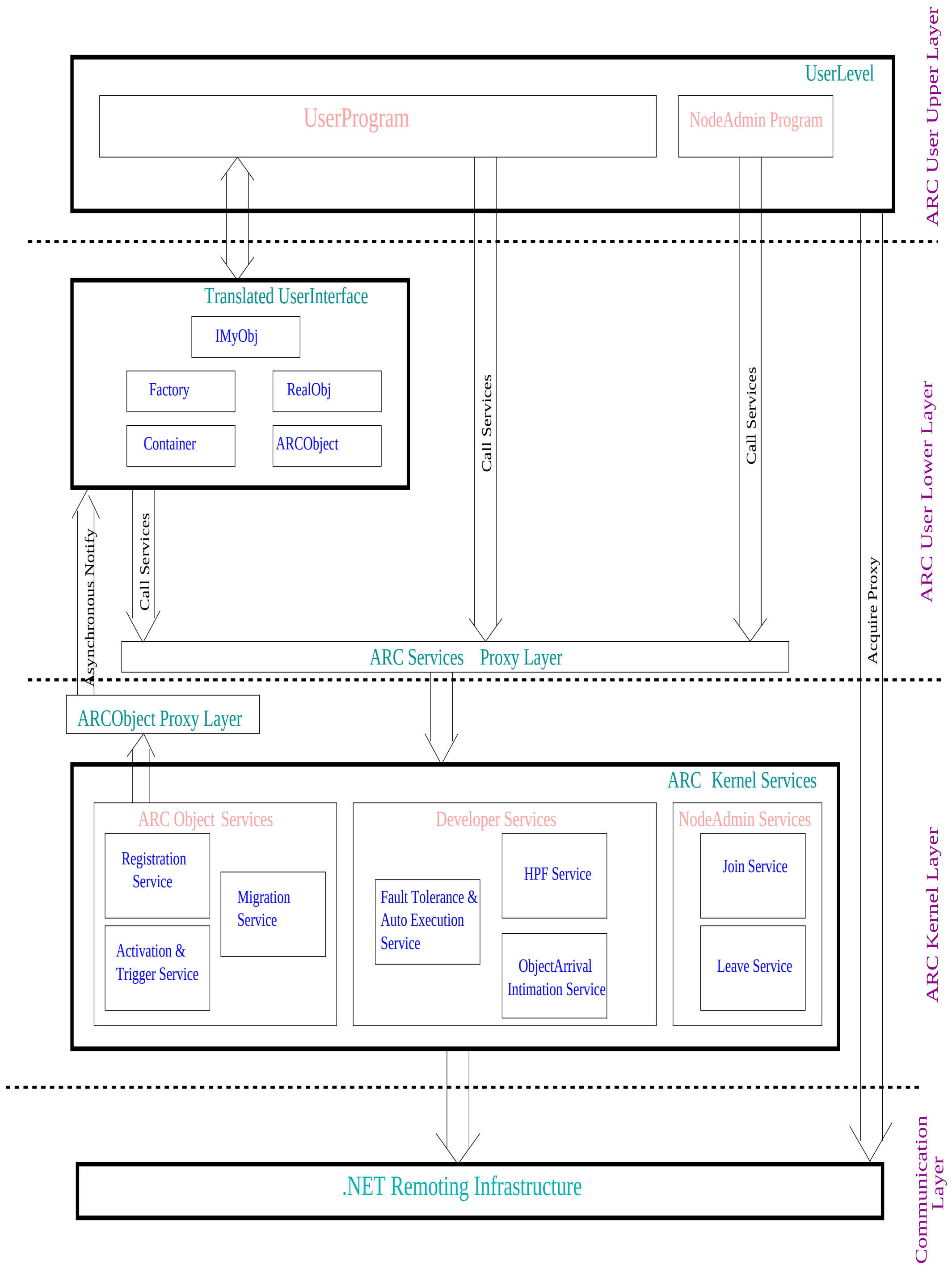

Figure 2: ARC Framework Architecture

3.4 Communication Layer

.NET provides remoting infrastructure which allows method invocation on a remote machine. Communication between any two machines takes place through this level. The .NET services accessed in this layer consists of registration of services and creation of proxies.

4 Application Development Process

An ARC application may be classified into broad categories of distributed, parallel and mobility based applications. An application may also combine one or more of these features. Applications may involve autonomous objects migrating over the network and exchanging messages. These distributed objects carry unique identities and may work co-operatively to serve a common purpose.

In an application involving parallelism, a large computation may be divided into small individual subcomputations expressed as ARC objects. Using ARC constructs, these subcomputations may be executed on remote machines in parallel. Results of remote executions arrived at the originator asynchronously as ARC objects retract. Earlier work on parallelism on an ARC platform is discussed in [4]. An application may also employ multiple hoppings of an ARC object to complete its work-flow. ARC system allows ARC objects to be migrated over a specific path.

This section highlights a development process of ARC programs through an example. In the example discussed, an ARC object is migrated to a remote anonymous machine, where a specified task is triggered on arrival. The object retracts asynchronously to originator context after completion of the triggered task.

Interface Specification

The development process begins with a specification of an ARC object interface as shown below. The interface consists of public member functions that the ARC object exports. These member functions are exported in addition to default ARC object members such as a trigger executor. In the interface specification below, the ARC object interface inherits a library interface $IRefCount$ to add reference counting feature to ARC objects of type $IMyObject$.

```
public interface IMyObject : IRefCount { // User specified interface
  void task(); // a public member of the ARC object
}
```

Implementation of ARC Object

Implementation of the interface is done in class *Real* in the namespace corresponding to user specified ARC interface. Skeleton of this class is generated from the ARC object interface. The architecture of the generated classes is discussed in next section.

```
namespace NSIMyObject{
   [Serializable]
   public class Real: PReal{
      public override void Trigger(){
         Console.WriteLine(‘‘This Message is Expected
                              to be Displayed at Remote Node’’);
         this.task(); //method to print Hello World!
      }
      public override void OnReturn(){ }
      public override void OnRetract(){ }
      public override void task(){
         Console.WriteLine(‘‘Hello World!’’);
      }
   }
}
```

During implementation of *Real*, definitions of member functions need to be stuffed in by the programmer. Method $Trigger()$ is automatically executed after the object migrates. In this example, method *task()* is called from within *Trigger()*. The code for class *Real*, which implements interface *IMyObject* is shown below. Notice that methods *Trigger()*, *OnReturn()* and *OnRetract()* are due to inheritance from interface *ITrigger*. The methods in interfaces $ITrigger$ and $IMyObject$ are to be implemented class *Real*.

Originator Application

Below is an example originator code. The originator program invokes a creation request on the factory class available under the generated namespace corresponding to the ARC object interface. After creation, the program sends instance of class *Real* to an anonymous remote machine and executes a local method in parallel. Finally the program blocks through a call to method *Sync()* till the migrated ARC object retracts.

```
namespace testHello{
  public class HelloClass{
    public static void Main(){
    // 1. Connect to local HPFServer
       UserInterface_HPFVector.IUser hpfvector;
       UserInterface_HPFServer.IUser hpfserver =
            (UserInterface_HPFServer.IUser)Activator.GetObject(
                        typeof(UserInterface_HPFServer.IUser),
                        "tcp://localhost:8105/HPFServerClass" );
   // 2. get HPFVector
       hpfvector = hpfserver.getHPFVector(1);
       int i  = hpfvector.SizeOfHPFVector();
       UserInterface_HPFValue.IUser hpf1=null;
       hpf1 = hpfvector.getHPFValue(0);
   // 3. Instantiate an ARCObject
       NSIMyObject.IContainer arcobject = NSIMyObject.Factory.New();
   // 4. send created object to remote machine
       arcobject.push(hpf1);
   // 5. do any work in parallel
       Console.WriteLine(‘‘to be executed in parallel’’);
   // 6. wait for object to come back
       arcobject.sync();
   // 7. end of program
       Console.WriteLine(‘‘the end’’);
    }
  }
}
```

Note that the HPF value obtained through the HPF service represent a machine on which the ARC object migrates. The remote machine remains anonymous to the originator application program. After executing a predefined task in its trigger specification the object retracts. In the meanwhile the originator performs an activity in parallel.

5 Summary

Architecture of a framework for Anonymous Remote Computing for C# over .NET was discussed with an emphasis on higher layers of the framework. The framework itself is designed using object oriented methodology. It supports the three features of parallelism, distribution and mobility including multiple hopping and connectivity in presence of mobility. The paper also highlighted the mechanisms for application development and implementation of the higher layer of the ARC .NET framework. A method of application development was also discussed.

References

[1] Rushikesh K. Joshi Aruna. L, Yamini Sharma. Object-centric Design of an ARC Kernel. In *Proceedings of HPCN*, volume LNCS 2110, pages 251–262, 2001.

[2] Grady Booch James Rumbaugh, Ivar Jacobson. *The Unified Modeling Language Reference Manual.* Addison Wesley, 1999.

[3] Dale Rogerson. *Inside COM*. Microsoft Press, 1997.

[4] D. Janaki Ram Rushikesh K Joshi. Anonymous Remote Computing, A paradigm for Parallel Programming on interconnected Workstations. *IEEE Transactions on Software Engineering*, pages 75–90, Jan/Feb 1999.