A Scheme for Implementing Ambient Calculus over an ARC Framework

Position Paper

E. Vasudeva, Rushikesh K. Joshi
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Email: {vasudeva, rkj}@cse.iitb.ac.in

Abstract

Ambient Calculus can model mobile agents across administrative domains, which may be distributed over the Internet. In this paper, we discuss a scheme for supporting ambients over Anonymous Remote Computing (ARC) framework for C# over .NET. The service oriented framework supports parallel and distributed computations in presence of mobility. A scheme for representing ambients in C# is discussed and a scheme for implementation over the ARC framework is proposed.

1 Introduction

Ambient Calculus was proposed by Gordan and Cardelli [3]. In this section, we take a brief overview of the constructs of ambient calculus which are in the scope of the scheme presented in Section 3.

An Ambient is defined abstractly as a bounded place where computation occurs. In an implementation, it may represent a enclosing program, collection of objects etc, for which the boundary is well defined. Thus, an important property of an Ambient is that there exists a clear boundary that defines the content of an Ambient. The reason is that when an ambient needs to be moved, it should be clearly known what should move and what should not be. The boundary is used in moving the ambient as a whole. Ambient Calculus separately captures constructs for mobility of ambients and communication between them.

Examples of ambients that define clear boundaries are individual web pages bounded by the enclosing file, an independent component bounded by its implementation class and a laptop bounded by its physical case and ports. An example of a non-ambient is a thread from a collection of threads sharing common data. An independent thread is not an ambient since threads share data. However, all threads along with the data that they share may form an ambient.

Ambients are identified by their names. An ambient 'n[P]' is an ambient with name n and process P. An ambient can contain sub-ambients and parallel processes as in n[m[P1]—P2—P3]. Processes running inside an ambient are called as control agents as they control the movement of the ambients through calls to mobility constructs *in* and *out*. The in, out and open calls are blocking calls and hence the processes get blocked until the required named Ambients are available at the intended place to finish the call. For example, 'out m.P' moves the calling ambient out of parent ambient m and continues with process P. The movement blocks till the named ambient is available as parent. In order to

Figure 1: Ambients over ARC

Figure 2: Class hierarchy

access an ambient, a capability to access it should be present. For example, for an expression 'in a.P' to complete its movement of calling ambient into ambient a, 'a' must be available as a sibling of the calling ambient. If more than one ambient are present with the same name, any one of them can be chosen [2]. An expression 'open n.P' opens ambient named n and continues with process P. The process that executes open has to be a sibling of n for open to be successful.

A prototype for non-distributed implementation of Ambient calculus is described in [1]. In this section, we will describe a high level design for implementing ambients on top of an ARC environment. ARC for C# on .NET [4] is a service oriented framework that supports parallel and distributed computations in presence of mobility. We can exploit the mobility features of the ARC framework to implement a distributed version of the ambient calculus. The proposed architecture is shown in Figure 1. For further details on ARC, the reader is referred to [4].

2 Ambient Implementation over ARC

The ambient runtime acts as the topmost ambient at a node, thus inherently is a default parent of all the ambients at that node. This default is overridden to form hierarchies. The ambient runtime decides when an ambient needs to be migrated to a remote node based on the calls and availability of the capability. If the ambient needs to be migrated, ambient runtime makes use of ARC framework to transport the running ambient to the destination host.

A class hierarchy for implementing Ambient Calculus in C# is shown in Figure 2. The high level C# code is shown below. Core to the implementation is a composite design pattern that allows composite ambients with processes. Abstract class AbstractAmbient is implemented either as a terminal node, that is a process, or as a composite Ambient which may have many ambients and processes within its scope.

```
abstract public class AbstractAmbient {
         public Ambient parent; //parent is always an ambient
         public AbstractAmbient(Ambient p) { parent=p; }
         abstract public void execute();
}
```

The argument to constructor is used to set the parent ambient for a newly created ambient. The default used is top ambient in a given distributed ARC based environment. Method execute() is implemented by subclasses of class AbstractAmbient. In the case of composite ambient, execute is dispatched to all its children and in the case of a process, the execute method is implemented as the body of the process by user process classes that inherit from class Process in the above hierarchy. Class Process is defined as below through inheritance.

```
abstract public class Process: AbstractAmbient {
       public Process(Ambient parent): base(parent) {
          // create a separate thread to execute the process
          parent.addChild(this);
          Thread t= new Thread( new ThreadStart(execute));
       }
}
```

As soon as a process is created, the execution of its logic is triggered as a new thread after its parent ambient is set. Actual code of execute() is provided by concrete user process classes.

Below code shows an implementation of composite class Ambient. A helper class Name to implement names of ambients is also provided. Class Ambient implements mobility primitives open, inamb and outamb. These are invoked by processes through their parent ambients.

```
public class Name { .... } // implements ambient names
public class Ambient: AbstractAmbient, IAmbient {
// declare a Collection AbstractAmbients here
       Name name;
       public Ambient(Name n, Ambient parent):base(parent) {
              name=n;
              parent.addChild(this);
              Thread t= new Thread( new ThreadStart(execute));
       }
       public void inamb(Name n) {...}
       public void outofamb(Name n){...}
       public void open(Name n) {...}
       public void addChild(AbstractAmbient p) { ... }
       public override void execute() {
            //for all items in collection, call execute
            //thus running all the processes in parallel
       }
}
```

When a parent ambient receives a call inamb(n), it looks for a sibling with name n and moves into n. On receiving a call outamb(n), it looks for parent with name n. On receiving a call to open(n), the ambient looks for a child ambient named n and opens it up by changing the parent ambient of all the children of the chosen ambient n. ARC mobility services will be used in implementing these calls if target ambients are remote.

3 An Application

Using the above implementation, we describe an example of an ambient in which two processes synchronize, by using ambient constructs. The abstract ambient code for these examples can be found in [3]. In the code below, processes PLock and PRelease synchronize on an ambient lock named *lock*.

```
public class PLock: Process {
       public PLock(Ambient p): base(p){...} // create in parent p
       public override void execute() {
              Name n=new Name("lock");
              parent.open(n);
       }
}
public class PRelease: Process {
       public PRelease(Ambient p):base(p){}
       public override void execute() {
              Name n=new Name("lock");
              new Ambient(n, parent); // create as a sibling
       }
}
```

```
public class LockingAmbient : Ambient {
       Process p1,p2;
       public LockingAmbient(Name n,Ambient p): base(n,p) {
              p1= new PLock(this);
              p2=new PRelease(this);
       }
}
```

In a distributed implementation, a naming server may actually hold the names and locations of the topmost ambients at different hosts. Thus any transfer in between hosts is appropriately initiated, and the ARC layer transports a given ambient to the required remote destination host.

4 Summary and Future Work

We observed that no full fledged implementation of ambient exists for widely used programming language environments. We discussed a scheme to implement the calculus in C# and subsequently over ARC framework by exploiting the mobility features of ARC framework. At the core of the design is a composite hierarchy that integrates processes and ambients. Ambients may be implemented as moving ARC objects.

References

[1] Luca Cardelli. Mobile Ambient Synchronization. Technical Report SRC Tech Note 1997-013, 1997.

[2] Luca Cardelli. Abstractions for mobile computation. *Secure Internet Programming: Security Issues for Mobile and Distributed Objects. Lecture Notes in Computer Science*, 1603:51–94, 1999.

[3] Luca Cardelli and Andrew D. Gordon. Mobile ambients. *In Foundations of Software science and Computational Structures, LNCS 1378*, pages 140–155, 1998.

[4] T. Vamsi Kalyan and R.K. Joshi. Architecture of the object oriented anonymous remote computing framework for c# over .net. *Software Design and Architecture Workshop, Bangalore, India*, Jan 2004.