FarCOM™

Mobile Objects over DCOM

Technical Whitepaper
2000

Technion
Hafia, Israel

FarCOM Overview

FarCOM is a DCOM based middleware, enhancing the current DCOM model with mobile COM components. FarCOM gives the D/COM programmer the tools to create distributed, mobile, self-persistent COM objects using an easy, intuitive and familiar methods.

FarCOM is being developed as a final research project in the Software Lab, Electrical Engineering faculty, Israeli Technion. FarCOM is an NT based project, but we also research for its applicability to run on CE machines, supporting DCOM.

FarCOM is based on a project called FarGo, which was submitted as a doctorate project by Dr. Holder Ophir, and Dr. Ben-Shaul Issy as his guide. The FarGo project created a framework for creating mobile components using Java. FarCOM uses the same terminology that FarGo uses, and uses a similar model for objects mobility. FarCOM manages to prove, that what was possible using Java, and seemed to be impossible using C++, was made possible using the D/COM model.

FarCOM's mobile components have the ability to move between machines. This movement is completely transparent to the users of the object. This means that users can work with an object whether it is local or remote, without knowing its exact location, and not knowing whether the object has traveled while using it. In addition, the programmers that create the mobile objects need to put very little effort in programming the object, or making an already existing object a mobile one.

FarCOM Key Features

- Objects are mobile on a distributed environment
- Objects are uniquely named
- Objects are self persistent (self-serialization)
- Users can easily bind to existing objects
- Mobility is transparent to users
- Creating mobile objects uses ATL-based interface
- FarCOM is wrapped with high level classes for ease of use
- Ability to test distributed applications on a single machine
- Distributed garbage collection of dead objects

FarCOM Basic Terminology

Mobile objects are called *Complets*. Complets are COM objects that may, or may not have out-going references. The first object that is created (meaning the root of the objects graph) is called the *Anchor*. A Complet can (and probably will) have a state, which is specified by its member variables. In addition, a Complet might point to other Complets. When a Complet moves, its state moves with it.

FarCOM supports several *Movement Semantics*. A movement semantic specifies what should be done when a Complet points to other Complets, and that Complet moves. Say that Complet A points to Complet B, and then Complet A moves. The movement semantics are Link (leave Complet B in place, but keep the reference valid), Pull (move Complet B together to wherever Complet A is moving), Duplicate (create a new instance of Complet B in the new location of Complet A and point to it) and Stamp (find an already existing instance of Complet B in the new location and point to it).

Movement Semantics is stored using an object called *Relocator*. The Relocator derived objects implement the different behaviors, according to the semantics they support.
Complets move between *Cores*, not machines. A normal state is to have one Core per machine, and thus Complets can move between machines. However, it is easier to test and debug applications on single machine and then execute them on a real distributed system. In this case, one can put several Cores on the same machine, and move Complets between the Cores.

FarCOM Operation
As mentioned earlier FarCOM gives the ability to create mobile objects. In order to support transparent movement, the user of the object must not point directly to the object (if the object moves, the pointer becomes invalid). This introduces an object called Tracker. A Tracker is an object that tracks a Complet's movement. When the Complet moves, it notifies the Trackers that point to it, that it has moved, and specifies the new location. The Trackers then invalidate their pointers.
The user of a Complet points to a local Tracker. The local pointer remains valid even when the Complet moves. In addition, the Complet can be local or remote, a fact that the user does not care about.
When a Complet moves, it creates a trail of Trackers, so that it will be possible to locate the Complet from any one of the Cores it had traveled through. Each of these Trackers must be invalidated when the Complet moves.
In FarCOM, the Cores are the managers of, basically, everything. They are responsible to create Complets and Trackers, move Complets upon a user's request, locate and bind to existing Complets and more.

FarCOM URLs
Cores and Complets are identified using a web based URL, which has the following format: FarCOM://www.hostname.com/CoreName#CompletName. If the CompletName is dropped then we have Core location. Using this URL, Complets and Cores can be created, bound to or moved.
FarCOM implements two class, CoreURL and CompletURL, that wraps this URL and provides an easy-to-use interface to these URLs.

FarCOM Elements Implementation
Cores are implemented as out-of-process COM servers. They use custom class factory based on names (so there will be exactly one instance of a Core per unique name). They implement the IFCCore (stands for FarCOM Core) interface.
Trackers are implemented as in-process COM servers. They are derived (aggregated) from ICompletPtr interface, and implement the IFCTracker interface.
Complets can be any object that the programmer decides. They are provided as a template class which implement the IFCComplet interface and the IPersistStream interface.

Complet Movement
Complets do not really move in FarCOM. Rather, they are created on a new location and removed from the original location. The basic algorithm is:

- Test if the object can and notify it that it is about to move
- Save object's graph to stream according to movement semantics
- Create a new object in the target Core
- Release the original Complet, and free the memory it uses
- Construct a new object graph using the stream from the old one
- Invalidate the Trackers so they can point to the new object
- Notify the object that the movement has completed

Self-Serialization
The programmer of the Complet does not need to implement the Save and Load methods of the IPersistStream interface. This functionality is provided to him when he creates his object as a Complet (by deriving his class from IFCCompletImpl template class). The only effort that is required from the programmer it defining the state map of his implementation. This is done using ATL-style macros.
Using the state map and movement semantics FarCOM can dump the entire object graph to a stream and then reconstruct it in the target Core without bothering the programmer with the internal, complex code.

FarCOM Wrappers
In order to comply with FarGo, and provide an easy, intuitive way of using FarCOM, we added a set of static classes that wraps all of FarCOM's abilities. This way, even someone who understands very little about DCOM can use the system and create sophisticated distributed applications with mobile objects.

FarCOM keywords
FarCOM, FarGo, Distributed Applications, Mobility, Nomadic Objects, Mobile Objects, COM, DCOM, Class Factory, Serialization, Persistent Objects

Acknowledgments
Author
Weinsberg Udi (email: udi@tochna.technion.ac.il)
Developers
Weinsberg Udi, Elmaleh Liron
Consulting
Weinsberg Yaron, Ben-Shaul Issy, David Ilana
Based on work by
Holder Ophir, Ben-Shaul Issy, Hovav Gazit, Gidron Yoad, Weinsberg Yaron, Abu Micky