Nova - Component-Oriented Application Development

Andrew Nurse, openarrow@gmail.com

February 9, 2005

1 Introduction

1.1 Work In Progress

This document is a work in progress. Anytime you see "**??**" as a section reference, it refers to a section which does not yet exist. Please contact the author at "openarrow@gmail.com" if you would like to contribute or help in this project. Please report errors in this document to e-mail to the address above.

This document was created from a LaTeX2e input file and can be converted into DVI, PS, PDF, or some other format using an appropriate LaTeX processor.

1.2 Assumptions

This document assumes you have some idea of how the CLR and Microsoft's implementation of it, in particular, work. You should also have some knowledge of the C# language as most of the code listings in this document are written in that language.

1.3 Code Listings

Throughout this document, sample code will be presented in the form of code listings. These listings are separated out from the document and coloured as they would be in most standard programmer text editors. All listings are in C# unless otherwise specified. See Listing 1.1 for a sample listing.

```
public void SampleMethod()
{
        // This is a sample listing
        Console.WriteLine("Sample Listing");
}
```

Listing 1.1: A Sample Listing

Component	**Object**
Contains multiple objects	Is one object
Designed to perform a small set of tasks	Represents an entity
Designed for portability	Tied to an application
Designed for transparent remoting	Remoting is implemented by other systems

Table 2.1: Differences between Components and Objects

2 Overview

2.1 What is a Component?

The word "component" has many meanings in software development. It can be an abstract concept of a single encapsulated entity that performs a certain task. Or it can be a more concrete concept such as the idea of a COM Component (see Section **??**, "Nova vs. The Component Object Model). Nova is an implementation of the eXtensible Application Development System (XADS, pronounced 'zads') which provides a specification for "Component-Oriented Development." COD is a further development of the concepts provided by Object-Oriented Programming (OOP) in that a component, as defined by the concepts of COD, makes use of multiple objects in its implementation. However, a component is very different from an object, and Table 2.1 summarizes these differences.

2.2 What is an Application?

Like a component, the term "application" also has many meanings. Usually it refers to a program that a user can execute to perform a certain task, but this is only a loose definition. Nova, and XADS, defines an application as a particular set of interactions between components as coordinated by an "Application Component." What does this mean? It means that an applications is simply a group of components working together, and this cooperation is

managed by a component that is specific to the application. If you are familiar with .Net, and specifically C# (see Section 1.2, "Assumptions") this Application Component is similar to the entry point to an application. In effect, the Application Component is loaded by Nova and sent a "Main" message (see Section 3.1, "Messaging") which is equivalent to calling the entry point method of a C# application. The component receives this message and begins sending messages to other components to get them working. Section **??**, "Application Startup Procedure" details this process.

2.3 Why should I care?

So, now you know what Nova considers a component and what it considers an application, but how does this affect you and your development process? This question is easy to answer with one simple statement: "Components are designed to be as portable as possible." This means that if you develop an application, you must try to develop all your components so that they are usable by any other application (with your permission of course, you could even charge for it) and in turn, your application can use other components, even though they are not developed for your application.

As an example, consider an archive management application (for managing zip, tar, etc. archives), this application has user interface components and management components to manage the interface. Now, that application needs a library to manage its main archive format (say FSA, the Freakin' Small Archive, a hypothetical format), you could write a component for that format and then sell it (or give it away free, or any number of other licensing options) and others could create FSA files easily. Also, you could acquire a ZIP file component and incorporate it into your application, allowing you to manage ZIP files as well. As a final step, you could use Messaging Contracts (see Section **??**, "Messaging Contracts") so that you can have your application, at runtime, search for archive format components and download them on the fly through Discovery Servers (see Section **??**, "Discovery Servers").

Clearly, Nova provides a great deal of advantages, but there are some costs. Component messaging is not as fast as object messaging (in most cases), because Object messaging is usually implemented as simple function calls. However, component messages are queued and prioritized and may be required to pass between multiple machines (see Section **??**, "Remoting"). Though contracts and disco servers are extremely useful, they usually require some sort of central registry to ensure two contracts do not perform the same task (causing incompatibility between them). Nova requires some infrastructure on the end user's computer in the form of Component Servers (see Section **??**, "Component Servers"). However, in many cases, the benefits

outweigh the costs and overall Nova improves application development.

3 Architecture

This section will provide a description of the architecture of the Nova system.

3.1 Messaging

One of the primary goals of Nova is to facilitate communication between components on the same machine, or across the world. This is achieved through a messaging system that is consistent even if the message is sent to a computer halfway around the world. At the core of this system is the message structure itself (see Section **??**, "Message Structure"), which defines the exact format of a message. Also, a system for message routing (see Section **??**, "Message Routing") allows messages to cross various application boundaries (see Section **??**, "Application Domains") seamlessly and without any extra action by the sender. Finally, a standardized serialization system (see Section **??**, "Message Serialization") is required to ensure messages can be easily sent over various transport protocols (see Section **??**, "Transport Protocols").

3.1.1 Message Structure

Nova uses messaging to allow components to request that other components perform actions or send them data. This process is very similar to the way object-oriented messaging works in that an OO message consists of an action to be performed (method name) and data to aid in the performing of that action (parameters). These two attributes (action and parameters), along with the component name (see Section **??**, "Component Naming") of the sender and the reciever, make up the main data contained in the message (other information is discussed in later sections). In Nova, the arguments to a message are .Net `object` instances which are serialized by various transport protocols as defined in Section **??**, "Message Serialization."