ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

Internalization of Dynamic Slicing in Delegate Multicast Events for E-tracking Systems in C#

Divanshi Priyadarshni Wangoo
ITM University, Sector-23A, Gurgaon, Haryana, India
divanshi.wangoo@gmail.com

Abstract
This paper focuses on the construction of SDG for delegates called DDG which will make message passing and event handling in delegates simpler. An Event is an important C# feature that lays its foundation on delegates. The DDG can be incorporated into .Net platforms and slices can be computed dynamically at the time of occurrence of an event. Generating events and passing the action of the triggered event to the event handler can be dynamically generated at run time with the assimilation of DDG into the event handling mechanism. "E-tracking systems" can have better trailing if the DDG is embedded into the event handling application programs in .Net platforms. Multi-cast events enable multiple objects to respond to an event simultaneously and give flexibility to the user to get response of more than one action on the triggered event. This brings efficiency to the programmer and helps in better maintenance of the software application project and also helps in reducing the execution time.

Keywords
System Dependence Graphs (SDG), Delegate Dependence Graphs (DDG), Event handling, Dynamic slicing, Graphical User Interface (GUI), Integrated Development Environment (IDE)

I. Introduction

C++ developers do not have access to Windows Forms as a forms generator. They are restricted to coding the graphical user interface (GUI) from managed classes found in the Windows.Forms namespace. There is no visual assistance. C-based programmers must use C# for full access to Windows Forms visual tools.

C# opens a new door by including the feature of Event Driven programming such as Events and Delegates. Events are the means by which Windows Application receive automated notification of the action that has occurred. In a Windows application a lot of Events are occurring at a particular instant for e.g. Mouse Move, Mouse out, Mouse Click etc. Delegates are pointer to the function and are type-safe. Delegates are best complemented as new type of Object in C#. They are also represented as pointer to functions. Technically, delegate is a reference type used to encapsulate a method with a specific signature and return type i.e., a delegate can be used to call any method that matches the signature and return type as that of the delegate. If we consider a real world scenario then delegates can be understood as any delegate representing a country a group of people representing a company etc. This same definition can be mapped to C# as delegate act as an intermediary between event source and destination. The .Net Framework has a Namespace System which is a top-level namespace found in the class library of the .NET framework. Delegates are derived from System.Delegate namespaces. Delegates are useful for two main reasons. First delegates support events. Second, delegates give your program a way to execute methods at runtime without having to know precisely what those methods are at compile time. This ability is quite useful when you want to create a framework that allows components to be plugged in [2]. We have two flavors of delegate in C#; Single Delegate, Multi-cast Delegate. A delegate is called a single delegate that derives from the System and Delegate class contains an invocation list with one method. A delegate is called Multi-cast Delegate that derives from the System and Multicast Delegate contains an invocation list with multiple methods

In Multi-casting we create a single delegate that will invoke multiple encapsulated methods. The return type of all the delegates should be same. Now the question arises; why to use Multi-cast delegates when Single-cast delegates are enough? Well the answer to this question is in the need to call three methods when a button is clicked. The Multi-cast delegates are used with events where multiple calls to different methods are required. Multicast events can be used with multicast delegates when multiple objects respond to the same event message at run time.

A. Object-oriented Slicing in .Net(DotNet) Framework Platforms

The .NET framework was created by Microsoft as a software development platform. It was designed to accommodate Rapid Application Development (RAD), platform independence and network transparency. .NET is a multi language and multiplatform operating environment in contrast to Java, which is single-language and multi platform. .NET offers C#, Visual Basic .NET, and many more .NET-compliant languages. Programming in .NET does not require learning an entirely new language. To program to the Java Virtual Machine (JVM) requires learning the Java language.

.NET applications are build in Windows and run in Linux, UNIX, Macintosh, or any platform that offers a common language runtime (CLR). .NET introduces a new component model that is largely implicit. The messiness of COM (Component Object Model) is removed. In .NET, developers use standard language syntax to create, publish, and export components. There is nothing else to learn. .NET addresses many of the shortfalls of COM, including susceptibility to DLL. .Net provides a reflective, object–oriented API and it is designed for generic application, so that it can accommodate numerous programming and development languages.

Dot NET framework offers relatively strong integration with the Windows operating system and the Component Object Model (COM). The support and control backed by SQL Server gives strong back-end support & help the firms for improved site management.

.NET is tiered, modular, and hierarchal. Each tier of the .NET Framework is a layer of abstraction. .NET languages are the top tier and the most abstracted level. The common language runtime is the bottom tier, the least abstracted, and closest to the native environment. This is important since the CLR works closely with the operating environment to manage .NET applications. .NET supports managed and unmanaged programming languages.

1. E-tracking Systems in C#

Although researches have been done in the area of program slicing in various object-oriented programming platforms, there remains a scope of future extension to the use of slicing in unique generation of events at run time. The application and usability of such new technique especially for delegates and events can be accessed from the "e-tracking postal delivery" or courier tracking system. The user generates more than one event like determining the date & time of the item delivered and at the same time wants to check the status of another item separately through all the access points from the source to the destination through the use of interfaces like button, links, etc. The programmer can easily switch between the events that the user generates dynamically and simultaneously at a particular time. When the user clicks on multiple interfaces that are the GUI's programmed in the C# window forms, the event handler would be called only by the event to which it is triggered. With the use of dynamic slicing in multicast events delegate programs, multiple event generation and handling becomes simpler and error-free. Event handling being an integral part of the delegate application becomes vital for the dynamic run time execution of the event-driven program. A program slice for events in delegates would comprise of a slice of an event that the user wants to generate dynamically with respect to the delegate type.

B. System Dependence Graphs in C#

System Dependence Graphs (SDG) as in [1] is a graphical approach for calculating the dynamic slices based on a particular execution trace and slicing criterion. Although researches have been done in the construction of SDG's for C, C++ and Java programming platforms as in [2], there remains a scope for a constructive developmental approach in the field of .Net (Dot Net) based programming languages like C#, Visual Basic, Jscript, etc.

C# programming language is native to .Net based platforms and is a pure object-oriented language like Java [3]. C# is a future multi-paradigm language derived from C or C++ family and is highly suitable for building web-based applications that require ease of coding and flexibility.

Computing dynamic slices at the time when event is to be generated can be determined from the application which can handle the application of event with delegates.

1. Delegate Dependence Graphs (DDG) for Message Passing & Callback Methods in C#

There is a need for creation of delegate when an object holds reference to a method and the method is called using this reference [4]. With the use of delegates the method invocation depends on the references to the methods stored by the delegates i.e., a delegate can hold reference to more than one method and the method calling is determined at runtime rather than compile time. Delegates are also used for callback method mechanism.

A Delegate Dependence Graph (DDG) is a directed System Dependence Graph (SDG) where the nodes represent program statements and the directed lines represent the dependency between the nodes. Directed straight lines represent control dependencies, directed dashed lines represent data dependencies and directed dotted dashed lines represent the invocation of delegates (events) and dotted lines represent delegate instantiation. The DDG is constructed for the program in Fig. 1 using delegates with respective conventions as cited in Fig. 2.

```
Using System;
1.      delegate int delegateop(int a, int b);
2.    class DelegateDemo
{
3.      public static int Add(int x, int y)
{
4.      return(x+y);
}
5.      public static int Sub(int x, int y)
{
6.      return(x-y);
}  }
7.      class TestingDelegate
{
8.      public static void Main()
{
9.      delegateop obj1=new delegateop(DelegateDemo.
Add);
10.     delegateop obj2=new delegateop(DelegateDemo.Sub);
11.     int result1=obj1(200,300);
12.     int result2=obj2(200,300);
13.     Console.WriteLine("result1="+result1 );
14.     Console.WriteLine("result2="+result2 );
}
}
```

Fig.1: Source Code for Delegate Creation and Implementation

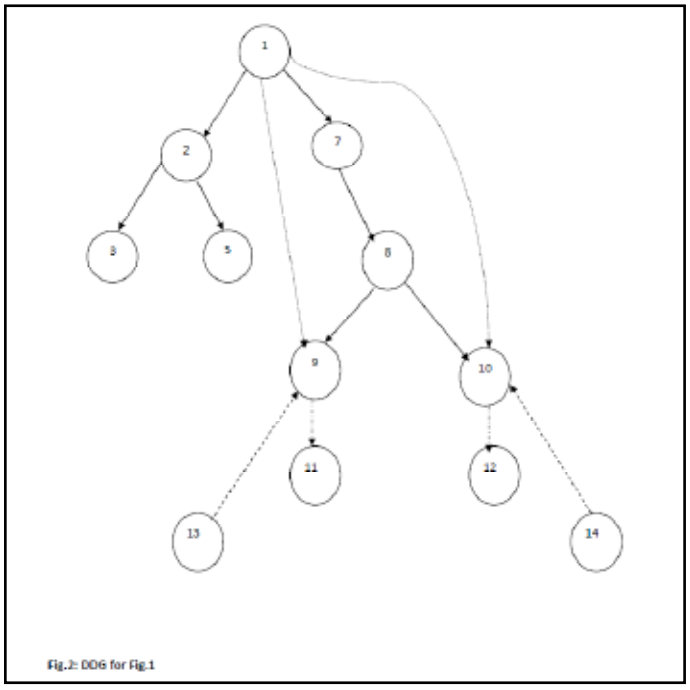

Fig.2: DDG for Fig.1

2. Multi-Cast Event Handling in Delegates Using Dynamic Slicing

An event is a notification that when occur depicts an action which is handled by the event handler of the particular event. Dynamic slicing can be computed for events generated by user at runtime by taking the slices of the events triggered and their respective event handler slices. Slices are taken at the point of action by the event handler i.e., when a user wants to trigger an event particularly at the run time execution of the program. For example, when the user wants to trigger clock events and raise an event for every new second, then the point at which the event is triggered to the

ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

point in the program where event handler is located is determined by computing the two slices at these two respective points. Thus, the event generation and handling mechanism becomes easier and the event handler method is called by only those events to which it is triggered.

An event is a delegate type class member that is used by the object or class to provide a notification to other objects that an event has occurred and the client object who wants to act on an event can do by adding an event handler to the event [3]. So generation of events at the run time dynamically can be made simpler by enhancing the DDG into the events based application in delegates. The programmer can easily depict and handle the dynamically generated events by computing the event slices for the event which is invoked at the execution time and slices for their respective event handlers. For time sensitive applications a user has to generate multiple events together at the same execution time . It applies to multi-cast events which enables multiple objects to respond to an event notification. Through multicasting of events a user can trigger the same event several times in the program. The event handler would be generated separately for each instance and added to the event chain. The dynamic slices for the event would consist of all the statements in the programs that are related with the triggering and handling of the event. This can be further depicted from the application of delegates to event handling. Its further extension has a practical relevance to the "e-tracking" application available to the users online.

The multi-cast event handling program and the DDG for it is illustrated in Fig. 3 and Fig. 4, respectively.

The DDG takes the slice points called event slice points at nodes 23 and 26 to call the event which is located at node 4 dynamically. Nodes 8 & 11 are the event handlers which are added when event is triggered at the nodes 21 and 22. The Event handler slice points in DDG at nodes 8 & 11 represent the handlers of the events which will be called at the nodes 21 and 22. Event slice point in the Fig.4 is the slicing criterion taken at the time of firing of the event and event handler slice point is the stream point where the event will flow after the event handlers are added to the list.

Therefore, with the automatic generation of these slice points in the .NET IDE, multiple event generation has a potential existence. Thus, multiple events can be generated by the user with the consideration of event slicing point & event handler slicing point in the DDG. This will lead to more proficient addressing of the events to their respective event handlers. Directed lines represent control dependencies, directed dotted dashed lines represent the bindings of the events to their invocation and dashed lines represent the instantiation of the classes containing the events and event handlers.

```
Using System;
1. delegate void MyEventHandler();
2.class MyEvent
{
3. public event MyEventHandler SomeEvent;
4.public void OnSomeEvent()
{
5. if(SomeEvent != null)
6. SomeEvent();
} }
7. class A
 {
8. public void Ahandler()
 {
9. Console.WriteLine("Event received by X object");
} }
10. class B
 {
11. public void Bhandler() {
12. Console.WriteLine("Event received by Y object");
} }
13. class EventDemo2
 {
14. static void Handler()
{
15. Console.WriteLine("Event received by EventDemo");
}
16. static void Main() {
17. MyEvent evt = new MyEvent();
18. A aOb = new A();
19. B bOb = new B();
20. evt.SomeEvent += Handler;
21. evt.SomeEvent += aOb.Ahandler;
22. evt.SomeEvent += bOb.Bhandler;
23. evt.OnSomeEvent();
24. Console.WriteLine();
25. evt.SomeEvent -= aOb.Ahandler;
26. evt.OnSomeEvent();
} }
```

Fig. 3: Multicast Event Handling Program

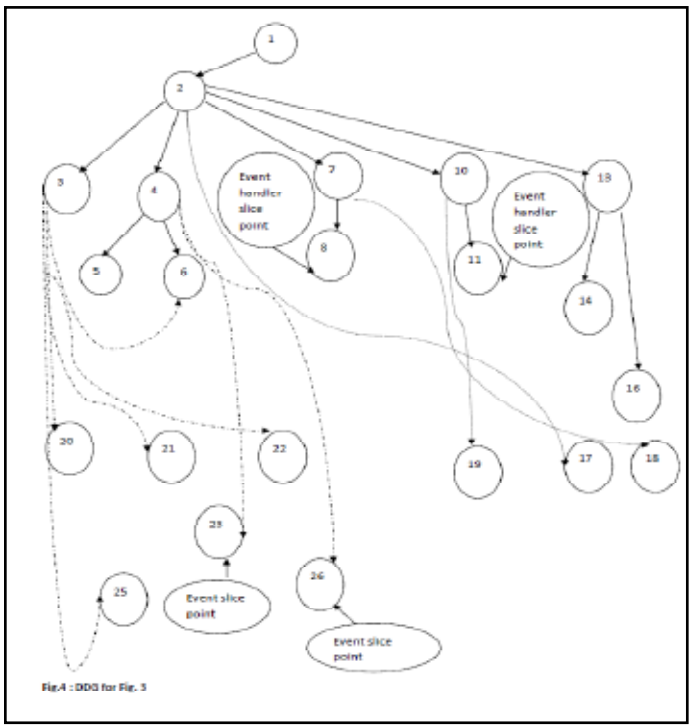

Fig.4 : DDG for Fig. 3

3. Applications of Event Handling

Event handling & catching method can be better explained with its application in "e-tracking segments" like online postal service. Computing dynamic slices at the time when event is to be generated can be determined from the application which can handle the applicability of the event with delegates. By taking the event slice points in the DDG built upon the programming application of "e-tracking systems", the user would be able to generate multiple events by clicking on the same event several

times. In the DDG of Fig. 4, the multiple event generation would occur when the event would be called several times at the event slice points & the handler would be called at the event handler slice points. The event slice points are the slices taken for the event generation dynamically at run time and event handler slice points are the peaks where the events will be handled. Thus, when the user clicks on the GUI elements on the e-tracking web sites, the event would be triggered and catched by the respective handler for that event. With the internalization of DDG into the programming application environments at run time would help to generate multiple events for the same interface. The event slice points and respective event handler slice points in DDG would be automatically taken by the .NET IDE compiler and would be activated at the time of user action. This will reduce the execution time and event handling would become less complex for the software programmer. The flow of control on event invocation through GUI can be depicted from
Fig. 5 below. The event slicing becomes activated when the action is passed to the event handler when the event is triggered. Thus the same flow of control will follow-up for all the events generated by the user and event slicing would be dynamically automated.

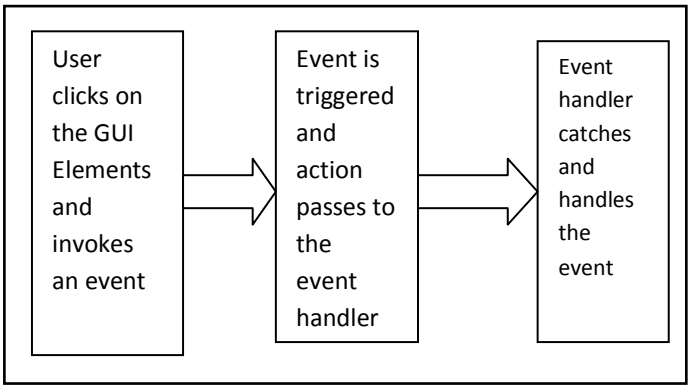

Fig. 5: Flow of Control on Event Invocation Through GUI

II. Implementation Results and Analysis

The support of dynamic slicing in event driven programming applications is riveting. Its internalization in the "e-tracking system" apparently acts as a supplement to the process of online services and to the end users. Its incorporation into the .NET IDE's will lead the software developer to an apparent horizon where the developer and application requirement converge to an estimable outcome.

III. Conclusion

The incorporation of dynamic slicing and DDG in multicast and single cast events built on delegates makes event handling more dynamic. It has a pragmatic approach to the e-tracking applications. The user can initiate multiple events dynamically with different objects for the same event. User communication through GUI and the flow of control to the event handling process becomes fast and execution time expedites. Multiple tasks can be done by the user at the same time leading to program efficiency. The software application can also be easily mend with dynamic changes. Thus the programmer has an ease to the maintenance of the software project. Time and cost is economized for the software companies at the hands of the developer.

IV. Acknowledgement

I would like to gratefully and sincerely thank my guide Dr. Supriya Panda of CSE &IT at ITMU for her guidance, understanding, patience and paramount mentorship in providing a well rounded steering for the achievement of my long term career goals.

V. Future Work

Future work includes extending the DDG for multicast delegates. Its advancement to generation of window forms where event handling becomes essentially vital. It can also be employed in time critical applications requiring accurate generation of events.

References

[1] *Donglin Liang, Mary Jean Harrold,"Slicing Objects using System Dependence Graphs", International Conference on Software Maintenance,Washington, D.C, pp. 358-367, November 1998.*
[2] *Neil Walkinshaw, Marc Roper, Murray Wood ,"The Java System Dependence Graph", Conference on source code analysis and manipulation, pp. 55-64, 2003.*
[3] *Balagurusamy, E, Programming in C#A Primer, 3rd edition, Tata Macgraw Hill, 2011*
[4] *Herbert Schildt,"The Complete Reference C# 3.0", McGraw Hill, 2009*