BECKHOFF

Verwenden von verteilten Ereignissen mit IBAEventAggregator

Henning Rausch
CNC Applikation
Telefon: + 49 (0)5246 963 5117
Telefax: + 49 (0)5246 963 9 5117
h.rausch@beckhoff.com

08.10.2013 **V 1.0 (MGehle)**

Mit Hilfe der Schnittstelle IBAEventAggregator bzw. dessen Implementierung(en) ist es möglich Ereignisse in verteilten Anwendungen zu verwenden, ohne dass der Auslöser des jeweiligen Ereignisses im Clientcode bekannt zu sein hat. Dazu wird das Publisher-/Subscriber- Pattern verwendet. Dieser Ansatz bewegt sich also abseits des Standard- .NET- Ereignismechanismus.

Die Umsetzung folgt weitestgehend der EventAggregator Klasse aus Prism (auch bekannt als „Composite Application Guidance for WPF & Silverlight“[1]).
Leichte Änderungen betreffen die Klassenarchitektur, sowie die Unterstützung für Windows Forms.

Die folgende Beschreibung geht nicht auf die einzelnen Schnittstellen und ihre Implementierung ein. Dazu sei auf die XML-Kommentare im Code verwiesen, vielmehr soll die richtige Verwendung dieses Eventmechanismus erläutert werden.

Notwendige Namespaces:

- Schnittstellen: `Beckhoff.App.Core.Interfaces`
- Basisimplementierungen: `Beckhoff.App.Core.Common.`

Grundlegendes

Bei der Entwicklung von Anwendungen, die aus mehreren unabhängig voneinander erstellten Komponenten bestehen, ist es oftmals wünschenswert, dass Komponente A über Zustands-änderungen von Komponente B (und umgekehrt) informiert wird.
Die Verwendung des .NET Ereignismechanismus setzt dazu voraus, dass Komponente A die Instanz von Komponente B kennt, so dass sie eine entsprechende Ereignismethode für das Ereignis von Komponente B registrieren kann.
Dieser Ansatz führt zu einer starren Kopplung zwischen den Komponenten A und B. Der hier vorgestellte Mechanismus hebt diese Kopplung auf, indem Ereignismethoden nicht bei dem Auslöser des Ereignisses, sondern bei einer zentralen Instanz einer Ereignisverwaltung registriert werden. Ein Ereignis muss entsprechend über diese Verwaltungskomponente auch ausgelöst werden.

Definition von Ereignissen

Ein Ereignis, was über `IBAEventAggregator` publiziert werden soll, muss stets die Schnittstelle `IBAEventBase` implementieren. In dieser Schnittstelle sind die grundlegenden Methoden, die auf das Ereignis angewendet werden können festgelegt. Anstatt jedoch `IBAEventBase` direkt zu implementieren, sollte die konkrete Ereignisklasse jedoch eher von `BAEventBase` erben. Diese abstrakte Klasse implementiert zum einen `IBAEventBase`, zum anderen bringt sie den notwendigen

[1] Siehe dazu auch http://compositewpf.codeplex.com/

Infrastrukturcode zur Verwaltung des Ereignisses mit.

Für den Event gibt es einen generische Klasse BAEvent<TPayLoad> hier können generische Typparameter übergeben werden. Dieser spezifiziert den Typen des Ereignisparameters, der vom Ereignisauslöser dem Ereignis beigefügt und damit von der Ereignismethode verarbeitet werden kann.

```
public class CncStateChanged : BAEvent<int> {}
```

In diesem Beispiel wird demnach ein Ereignis `CncModeChanged` definiert, welches als Parameter einen simplen Integer transportiert. Die Verwendung anderer Typen oder Klassen ist aber auch möglich. Die Ereignisklasse kann natürlich durch weitere Methoden oder Eigenschaften weiter aufgepeppt werden, für die Verwendung mit `IBAEventAggregator` ist diese Minimaldefinition jedoch schon vollkommen ausreichend.

Ereignisverwaltung

Die o.g. zentrale Ereignisverwaltung übernimmt eine Implementierung der Schnittstelle `IBAEventAggregator`. Diese Schnittstelle beinhaltet lediglich eine Methode namens `GetEvent`. Die Definition dieser Methode sieht wie folgt aus:

```
TEventType GetEvent<TEventType>() where TEventType : IBAEventBase, new();
```

Es wird hier also die Instanz einer Klasse zurück geliefert, die `IBAEventBase` implementiert.

Als Grundimplementierung von `IBAEventAggregator` steht die Klasse `BAEventAggregator` im Namespace `Beckhoff.App.Core.Common` zur Verfügung. Diese verwaltet intern eine generische Liste, von Singletons der angeforderten `IBAEventBase` – Implementierungen.

Veröffentlichen von Ereignissen

Das Auslösen von Ereignissen führt nun direkt über die o.g. Methode `GetEvent`. Im Folgenden und allen weiteren Beispielen wird davon ausgegangen, dass im ganzen System auf dieselbe Instanz von `BAEventAggregator` zugegriffen werden kann. Diese Instanz heißt in allen Beispielen `aggregator`.

```
var ev = aggregator.GetEvent<CncStateChanged>();

int newState = 15;
ev.Publish(newState);
```

Zum Auslösen des Ereignisses muss nun einfach die Methode `Publish` der Ereignisklasse (geerbt von `BAEventBase`) aufgerufen werden. Der Typ des einzigen Arguments dieser Methode entspricht dem generischen Typparameter von `BAEventBase`. Im Falle von `CNCStateChanged` ist dies also `int`.

Verwerten von Ereignissen

Das Empfangen und Verarbeiten von Ereignissen führt wieder über die Methode `GetEvent` der Instanz von `BAEventAggregator`.

```
var ev = aggregator.GetEvent<CncStateChanged>();

ev.Subscribe(this.HandleCncStateChanged);
```

Dieses Mal wird jedoch die Methode `Subscribe` auf die erhaltene Ereignisklasse aufgerufen. Diese erwartet als Übergabeparameter die Angabe eines Delegaten vom Typ `System.Action<T>`.
Der generische Typparameter `<T>` entspricht dabei wieder dem generischen Typparameter des entsprechenden `BAEvent`-Erben.
Die durch den Delegaten angesprochene Methode hat also ein Argument vom Typ `int`.

```
private void HandleCncStateChanged(int newState) {}
```

Die Methode `Subscribe` ist mehrfach überladen:

```
public IBASubscriptionToken Subscribe(Action<T> action, BAThreadOption threadOption)
```

Bei dieser Variante wird neben dem Delegaten auch ein Wert aus der Aufzählung `BAThreadOption` erwartet. Die entsprechenden Werte haben folgende Bedeutung:

`BAThreadOption.PublisherThread`	Der Aufruf der Ereignismethode erfolgt im selben Thread, wie dessen Auslösung.
`BAThreadOption.UIThread`	TODO: Der Aufruf der Ereignismethode erfolgt im Thread, in dem die Methode Subscribe aufgerufen wurde (**Defaultwert**).
`BAThreadOption.BackgroundThread`	Der Aufruf der Ereignismethode erfolgt asynchron in einem BackgroundThread.

```
public IBASubscriptionToken Subscribe(Action<T> action,
                                      bool keepSubscriberReferenceAlive)
```

Standardmäßig speichert `BAEvent<T>` die Referenz der Ereignismethode in einer Instanz der Klasse `System.WeakReference`. Daher wird diese vom Garbage Collector erfasst, wenn nur noch `BAEvent<T>` diese referenziert.
Bei Angabe von `keepSubscriberReferenceAlive` = `true` wird eine harte Referenz verwendet, so dass sie vom Garbage Collector nicht erfasst wird, selbst wenn die übergeordnete Klasse längst zur Bereinigung frei gegeben wurde. In diesem Fall ist der Clientcode selbst für die Freigabe mittels der Methode `Unsubscribe` (s.u.) verantwortlich.

```
public virtual BASubscriptionToken Subscribe(Action<T> action,
                                             BAThreadOption threadOption,
                                             bool keepSubscriberReferenceAlive,
                                             Predicate<T> filter)
```

Zusätzlich zu den oben vorgestellten Varianten kann hier noch ein Delegat einer Filtermethode angegeben werden. Dabei kann es sich auch um einen Lambdaausdruck handeln. Damit ist es möglich die Ausführung der Ereignismethode an bestimmte Werte des Ereignisparameters zu binden. Im folgenden Beispiel wird die Ereignismethode `HandleCncStateChanged` nur ausgeführt, wenn der übergebene Ereignisparameter vom Typ int größer gleich 7 ist.

```
ev.Subscribe(this.HandleCncStateChanged,
             BAThreadOption.SubscriberThread,
             false,
             i => i >= 7);
```

Verwalten von Ereignismethoden

Die Methode `Subscribe` gibt eine Implementation der Schnittstelle BASubscriberToken zurück. Mit dieser lässt sich u.a. die Zuweisung von Ereignismethoden verwalten.

```
var ev = aggregator.GetEvent<CncStateChanged>();
var token = ev.Subscribe(this.HandleCncStateChanged);

if (ev.Contains(token))
 ev.Unsubscribe(token);
```

Im obigen Beispiel wird die Ereignismethode `HandleCncStateChanged` dem Ereignis `CncStateChanged` zugewiesen und das dabei erhaltene Token in der Variable `token` gespeichert. Über die Methode `Contains` wird nun geprüft, ob `token` eine Zuweisung zum `CncStateChangedEvent` darstellt. Da diese Bedingung zutrifft, wird anschließend die Zuweisung mittels der Methode `Unsubscribe` wieder gelöst. Damit wird die Methode `HandleCncStateChanged` bei Auslösung des Ereignisses `CncStateChanged` nicht mehr aufgerufen.

Standardereignis

Der oben vorgestellte Mechanismus lässt eine Vielzahl verschiedener Ereignisklassen zu. Um innerhalb der Beckhoff Applikation einem drohenden Wildwuchs entgegen zu wirken, wurde im Namespace `Beckhoff.App.Core.Common` das Standardereignis `BASimpleEvent` definiert. Die Klasse sieht wie folgt aus.

```
public class BASimpleEvent : BAEvent<KeyValuePair<IBAToken, object>>{}
```

Als Ereignisparameter wird hier ein Schlüssel/Wert – Paar genutzt. Als Schlüssel kommt eine Implementation von `IBAToken` zum Einsatz, während der Wert vom Typ `object`, und damit universell einsetzbar ist, wenn auch zu Lasten der Typsicherheit.