Dennie Bastiaan en Ilske Verburg

Een intelligente robot met Microsoft Robotics Studio

IT'S ALL ABOUT CHOICE

De intelligentie van een robot wordt voornamelijk bepaald door de complexiteit van het beslissingsmodel. Hoe reageert de robot efficiënt en intelligent op de input van zijn sensoren? Hoe coördineert hij zijn keuzes en waar geeft hij prioriteit aan? Problemen waarvoor Microsoft Robotics Studio een oplossing heeft. In dit artikel kijken de auteurs vooral naar de manier waarop de CCR ingezet kan worden in het beslissingsmodel van een robot.

Intelligente navigatie van een robot door een ruimte wordt voor een deel bepaald door de algoritmen die worden toegepast bij de navigatie. Maar wanneer sensorinput een belangrijke rol speelt in dit algoritme wordt een correcte en parallelle afhandeling van deze sensorinput minstens zo belangrijk, zo niet belangrijker. De CCR, de Concurrency and Coordination Runtime van Robotics Studio, speelt een belangrijke rol in de gelijktijdige afhandeling van sensorinformatie. De CCR is een library die twee taken heeft: concurrency oplossen met een transparant threadingmodel en de coördinatie van berichten die ontvangen worden van andere services zoals sensoren. Mark Oppedijk heeft de CCR al kort toegelicht in zijn introductieartikel over Microsoft Robotics Studio in het vorige .NET Magazine (#18).

Het proces van beslissen

De CCR is ontwikkeld door de Microsoft Research Group. De doelstelling hierbij was het realiseren van een library die de uitdagingen wegneemt bij het ontwikkelen met multi-threaded applicaties, zoals het starten, bijhouden, blokkeren en synchroniseren van verscheidene threads. Bij de ontwikkeling van Microsoft Robotics Studio bleek deze CCR een goede toevoeging op de toolset voor de parallelle afhandeling van sensorinput en activiteiten binnen de robotlogica. De CCR maakt gebruik van een port-based mechanisme. Een port (een FIFO-queue) gebruikt een handler voor de afhandeling van de berichten die in de port terechtkomen. Voor elke keer dat er een bericht in de port wordt gezet, zal een thread worden gecreëerd om de handler zijn werk te laten doen. De arbiters kunnen op verschillende manieren naar de port kijken. Ze bieden de mogelijkheid het berichtenverkeer op verschillende manieren af te handelen en daarmee aan te laten sluiten op de beslissingslogica die in een bepaalde situatie het meest van toepassing is. Deze verschillende arbiters zullen we in een aantal voorbeelden toelichten. Voor deze voorbeelden wordt gebruikgemaakt van een fictieve robot, uitgerust met een GPS-ontvanger en een webcam.

De robot op pad

Stel, je wilt met deze robot bewegende objecten detecteren in een omgeving terwijl je door deze omgeving navigeert. Ook wil je een blok dat zich in dezelfde omgeving bevindt, kunnen detecteren en de positie hiervan kunnen doorgeven. De GPS-ontvanger van de robot kun je voorzien van informatie over de positie van de robot en de webcam zorgt voor beeldinformatie uit de omgeving van de robot. De logica die je wilt gebruiken, is gebaseerd op de analyse van deze beelden uit de omgeving van de robot en de bijbehorende positie-informatie. Omdat de focus van dit artikel niet ligt op beeldherkenning van algoritmen, gaan we hier niet in op de exacte logica, maar bekijken we hoe de CCR gebruikt kan worden om het geheel te coördineren.
We laten zien hoe je bij het implementeren van deze logica gebruik kunt maken van verschillende typen arbiters die de CCR biedt voor de afhandeling van berichten. De Joined Receiver, één van de typen arbiters, gebruiken we voor het in één keer afhandelen van positie- en beeldinformatie. Bij deze arbiter wordt de delegate pas aangeroepen als de verschillende gedefinieerde berichten, in dit geval positie en beeld, zijn ontvangen. Daarnaast wordt de Multiple Item Receiver-arbiter gebruikt voor het verzamelen van verscheidene webcambeelden voordat de analyse van de beelden wordt gestart. Deze arbiter neemt pas actie als het gedefinieerde aantal berichten is ontvangen. In dit geval willen we vijf berichten van de webcam ontvangen hebben voordat we ze gaan analyseren.

Joined Receiver

Wanneer je streaming-informatie, zoals een webcam-beeld, wilt gebruiken in een robot voor de detectie van een blok, moet je deze informatie met regelmatige intervallen doorgeven. Aangezien wij hier positie- en beeldinformatie aan elkaar koppelen, gaan we ervan uit dat deze informatie met dezelfde regelmaat (bijvoorbeeld om de 200 milliseconden) wordt ontvangen en dus gezamenlijk afgehandeld kan worden. Deze gezamenlijke afhandeling kan gewaarborgd worden door het gebruik van de Joined Receiver, omdat deze pas actie onderneemt als er zowel een bericht met positie-informatie als een bericht met beeldinformatie is ontvangen.
In codevoorbeeld 1 is dit uitgewerkt. We definiëren hier in eerste instantie twee poorten om de berichten van de andere services, in dit geval de camera en de GPS, te ontvangen. In een normale Robotics-toepassing zou je de poorten vervolgens ook koppelen aan deze services. In dit voorbeeld laten we dat, vanwege de focus op de CCR, buiten beschouwing en vullen we de poorten zelf met berichten. Vervolgens definiëren we een handler en koppelen we de poorten aan deze handler via een arbiter. Zoals te zien is in het voorbeeld, vraagt een Joined Receiver van de handler dat deze beide typen van de poorten als argument heeft opgenomen. In de handler kunnen dan beide berichten worden gebruikt voor de afhandeling ervan. In dit geval gaat het om het koppelen van een webcam-beeld aan de positie.
In de constructor van codevoorbeeld 1 zie je verder een aantal dingen:

```
private Port<Position> _positionPort = new Port<Position>();
private Port<Image> _visualPort = new Port<Image>();

private void PositionAndVisualHandler(Position pos, Image img)
{
  if (FindItem(pos, img))
  {
    Console.WriteLine("Item found");
  }
}

public Program()
{
  DispatcherQueue dq = new DispatcherQueue();

  Arbiter.Activate(
    dq,
    Arbiter.JoinedReceive<Position, Image>(
      true, _positionPort, _visualPort,
      PositionAndVisualHandler));

  _positionPort.Post(new Position());
  _visualPort.Post(new Image());
}
```

Codevoorbeeld 1.

- Er wordt een DispatcherQueue gecreëerd, dit is de queue waar alle berichten, van welke poort en arbiter dan ook, uiteindelijk terechtkomen. De interne dispatcher van de CCR bepaalt dan welke prioriteit de taak krijgt, en of en hoeveel threads hiervoor worden aangemaakt.
- Arbiter.Activate activeert een nieuwe arbiter.

Er wordt een Joined Receive-arbiter gecreëerd die een Position- en een Image-bericht verwacht op de gespecificeerde poorten en is aan de handler gekoppeld.

Vervolgens wordt in beide poorten via de 'Post'-method een bericht geplaatst. Op het moment dat beide berichten ontvangen zijn, zorgt de arbiter er voor dat een taak in de DispatcherQueue wordt

```
private Port<Image> _visualAnalysisPort = new Port<Image>();

private void VisualAnalysisHandler(Image[] images)
{
  Console.WriteLine("Analyzing images...");
}

public Program()
{
  DispatcherQueue dq = new DispatcherQueue();

  Arbiter.Activate(
    dq,
    Arbiter.MultipleItemReceive<Image>(
      true, _visualAnalysisPort, 5,
      VisualAnalysisHandler));
}

private void PositionAndVisualHandler(Position pos, Image img)
{
  _visualAnalysisPort.Post(img);

  if (FindItem(pos, img))
  {
    Console.WriteLine("Item found");
  }
}
```

Codevoorbeeld 2.

```
Port<Image> _port;
Arbiter.Activate(
  dq,
  Arbiter.Receive<Image>(
  true, _port, ImageHandler));
```

Codevoorbeeld 3.

```
PortSet<Image, Exception> _portSet;
Arbiter.Activate(
  dq,
  Arbiter.Choice<Image, Exception>(
    _portSet,
    ImageHandler,
    ExceptionHandler));
```

Codevoorbeeld 4.

```
Arbiter.Activate(
  Arbiter.FromHandler(JustAHandler));
```

Codevoorbeeld 5.

geplaatst. In dit geval het uitvoeren van de PositionAndVisualHandler; het detecteren van het blok.

Multiple Item Receiver

Om bewegingen te kunnen herkennen op basis van visuele informatie en hiermee de mogelijkheid te creëren ze te ontwijken, is het nodig om bewegend beeld te analyseren. Zoals eerder genoemd, gaan we er van uit dat de beelden van de webcam periodiek worden verzonden. Met een Multiple Item Receiver kun je er voor zorgen dat pas wanneer vijf beelden zijn ontvangen, de arbiter de taak in de DispatcherQueue zet. Beeldanalyse voor het herkennen van een beweging wordt dus verricht op basis van vijf beelden.
In codevoorbeeld 2 is te zien dat hier een nieuwe poort voor wordt aangemaakt: _visualAnalysisPort. Voor deze port maken we ook een handler. Omdat een handler van een Multiple Item Receiver meer berichten afhandelt, verwacht de arbiter ook dat het argument van de handler een array van het port-type is. Zoals te zien is, wordt er een Multiple Item Receiver voor één port gecreëerd. Het derde argument geeft het aantal berichten aan, in dit geval vijf, dat in de poort moet zitten voordat de handler in werking wordt gesteld. Aangezien de PositionAndVisualHandler de berichten op de eerder gedefinieerde _visualPort al afhandelt, moeten de berichten worden doorgegeven aan de _visualAnalysisPort om er voor te zorgen dat ze ook door de VisualAnalysisHandler kunnen worden verwerkt. Hiervoor passen we de PositionAndVisualHandler aan. Deze post nu het image-bericht door aan de _visualAnalysisPort.

Door het gebruik van de Joined Receiver en de Multiple Item Receiver hebben we nu de afhandeling van gekoppelde informatie en het periodiek analyseren van een aantal beelden gerealiseerd. Uiteraard zijn er nog andere situaties waarbij gebruikgemaakt kan worden van deze arbiters. Denk bijvoorbeeld aan situaties waar je verscheidene sensorwaarden wilt ontvangen voordat je deze waarden wilt gebruiken voor een beslissing, zoals bij een temperatuurmeting of een evenwichtsorgaan in lopende robots.

Andere robotsituaties

In het voorbeeld dat we hier hebben gegeven, zijn twee van de mogelijke arbiters aan de orde geweest. De CCR beschikt nog over een aantal andere arbiters dat gebruikt kan worden bij de implementatie van het beslissingsmodel van een robot. Zo toont codevoorbeeld 3 de Receive-arbiter. De Receive-arbiter is de eenvoudigste arbiter. Wanneer één bericht van een bepaald type in de poort wordt geplaatst, gaat de handler werken. Ook kent de CCR de Choice-arbiter. Een Choice-arbiter werkt op een PortSet; een port waar meer typen berichten in gezet kunnen worden. De Choice-arbiter kan vervolgens op basis van het type bericht een andere handler aanroepen. Een veel voorkomende toepas-

sing hiervan is het afhandelen van onverwachte situaties zoals een exception. Een voorbeeld hiervan is te vinden in codevoorbeeld 4. De FromHandler-arbiter (codevoorbeeld 5) is een arbiter waarvoor geen port gedefinieerd hoeft te worden. De handler in deze arbiter zal per direct werken, maar dit zal, net als andere taken in de CCR, asynchroon uitgevoerd worden:

Andere toepassingen CCR

De CCR wordt geleverd als onderdeel van Microsoft Robotics Studio. In dit artikel hebben we ons zo veel mogelijk beperkt tot voorbeelden binnen roboticatoepassingen. Het is echter niet gezegd dat dit het enige toepassingsgebied is, integendeel. Met behulp van de CCR is asynchrone afhandeling een stuk gemakkelijker toe te passen door de complexiteit van threading voor een groot deel af te schermen en de ontwikkelaar een helder programmeermodel te bieden. De CCR kan dan ook prima gebruikt worden in andere toepassingen waarin asynchrone afhandeling is vereist en waarin je overweegt om threading te gebruiken.

De robot kan kiezen

Met de fictieve robot hebben we ons beperkt tot het gebruik van CCR om te laten zien welke mogelijkheden er zijn voor de coördinatie van berichten. We hebben laten zien hoe je met het gebruik van verschillende typen arbiters de afhandeling van berichten kunt sturen en de intelligentie van de robot kunt implementeren. Er valt voor een robot heel wat te kiezen.

Dennie Bastiaan is binnen het Working Tomorrow-programma van LogicaCMG (www.workingtomorrow.nl) afgestudeerd op het onderwerp Microsoft Robotics Studio. Voor vragen en opmerkingen is hij te bereiken via d.bastiaan@orange.nl.

Ilske Verburg is softwarearchitect bij LogicaCMG, www.logicacmg.com/nl. Voor vragen en opmerkingen is zij te bereiken via ilske.verburg@logicacmg.com.

Referenties

MSRS-site (webcasts, tutorials, downloads): http://www.microsoft.com/robotics
Microsoft Robotics Studio Wiki: http://channel9.msdn.com/wiki/default.aspx/Channel9.MSRoboticsStudio
CCR-user guide: http://msdn2.microsoft.com/en-us/library/bb483117.aspx
CCR in MSDN magazine: http://msdn.microsoft.com/msdnmag/issues/06/09/Con-