Verteilte Systeme Projekt

Dokumentation

Michael Sievers

690593

Konzept der Anwendung

Das Projekt beschäftigt sich im Allgemeinen mit den Problemen der Erweiterbarkeit in Runtime und dem verteilten Rechnen. In diesem Projekt wurde Microsofts WCF Technologie in C# mit dem MEF oder Managed Extensibility Framework verwendet. Diese Technologien wurden aufgrund ihrer robusten und erprobten Natur sowie dem gegebenen Feature Sets gewählt, welche gut zu den Problemen der Aufgabenstellung passen.

Aufbau

Die Solution namens VerteilteSysLib ist in 5 Projekte unterteilt:

Model enthält ein Interface IPlugin, welches von vielen der Projekte benötigt wird.

VerteilteSysHost enthält den Code, um die Services zu hosten und den zentralen Server zu starten

VerteilteSysNode enthält den Code um weitere Knoten, auf welche die Arbeit vom zentralen Host verteilt werden kann.

VerteilteSysClient enthält den Code um eine CLI Anwendung zu starten welche mit dem Server kommuniziert.
Der Client kann wie folgt bedient werden:

> Wenn man ein neues Plugin hochladen möchte:
> Upload [Enter] -> nun kann man den Pfad zur Plugin.dll angeben [Enter] -> Plugin ist Hochgeladen und wird als verfügbar angezeigt.
>
> Wenn man ein Plugin benutzen möchte:
> <Plugin name> <Parameter> [Enter] -> nun wird die Anfrage bearbeitet und das Ergebnis ausgegeben

WcfServiceLibrary1 enthält die Interfaces mit denen die Services kommunizieren sowie die Implementationen des PluginService und des NodeService. Des weiteren befindet sich hier der ServiceManager und das Interface für den NodeService.
Die Services die vom Host gehostet werden sind hier definiert.

Funktion

Das Interface IPluginService exposed 3 Funktionen als OperationContract:

```
DisplayAllPlugins()
UploadFile(byte[] data)
UsePlugin(string pluginName)
```

Diese Funktionen exposed der Service, sodass man vom Client auf sie zugreifen kann.
Ihre Funktionalität wird in PluginService implementiert.
Wird vom Client DisplayAllPlugins() aufgerufen, so wird die Anfrage durch den Endpoint and PluginService weitergeleitet, wo diese dann an den ServiceManager weitergeleitet werden. Der ServiceManager besitzt eine Liste mit allen Plugins, auf die er nun zugreift um ihre Namen weiterzuleiten, sodass der Client eine Liste aller verfügbarer Plugins erhält.

Wird vom Client ein Plugin.dll hochgeladen, so wird diese zuerst in ein ByteArray konvertiert, um anschließend über uploadFile() an den PluginService zu gelangen. Hier wird das ByteArray an den ServiceManager weitergeleitet. Im ServiceManager wird nun ein System interner Name für das Plugin generiert, im Verzeichnis ExternalPlugins, wird ein Verzeichnis mit diesem Namen angelegt und mit einem ByteWriter wird das ByteArray wieder zurück in eine dll mit dem System internen Namen konvertiert und dort abgelegt.

Das Plugin wird in seiner ByteArray Form mithilfe der Funktion sendToNodes() an alle Knoten geschickt. Was das mit den Knoten auf sich hat, wird später erläutert. Nun als dll in einem Verzeichnis im Host abgelegt, muss der Server nun nach neuen mit [Export(typeof(IPlugin))] Komponenten im neu angelegten Verzeichnis scannen um das Plugin in Runtime einzubinden. Hierzu gibt es im ServiceManager eine IEnumerable<Lazy<IPlugin>> plugins; hier werden die mit [ImportMany(AllowRecomposition = true)] erkannten Plugins gespeichert. Es werden alle richtig Annotierten Plugins erkannt die sich im Verzeichnis pluginDir in Zeile 29 befinden. **Dieser Pfad musste leider absolut angegeben werden, muss daher zum Testen geändert werden.** Wird nun das Verzeichnis eins neuen Plugins dem AggregateCatalog hinzugefügt und der CompositionContainer danach mit ComposeParts neu composed sind alle Plugins verfügbar.

Nun zu den Knoten. Die Knoten funktionieren mit dem Interface INodeService und INodeServiceCallback. Die Knoten sind Konsolen Applikationen die ihrem eigenen Service mit dem Host verbunden sind. Diese Verbindung ist eine Duplexverbindung, sodass Callbacks möglich sind. Die Funktionen der Nodes sind im Interface INodeServiceCallback definiert:
void SendPlugin(string name, byte[] plugin); ist dafür da, dass der Server ein Plugin mit Namen an die Node schicken kann.
string DoWork(string message); ist dafür da um der Node zu sagen welche Aufgabe sie machen soll, also welches Plugin mit welchem Parameter benutzt werden soll.
Diese Funktionen werden von der Node Implementiert, wo bei ankommendem Plugin analog zum Server das Plugin gespeichert und in Runtime implementiert wird.
Erhält der ServiceManager den Befehl ein Plugin auszuführen, fragt er zuerst das Plugin wie viele Nodes es zum bearbeiten dieser Anfrage benötigt und wie der Parameter auf diese aufgeteilt werden kann. Er holt sich dann dementsprechend verfügbare Nodes. Diese haben sich bei ihrem Start mit registerNode() und ihrer Id beim Server angemeldet, wo nun Ihre Identität in Form ihres Callbacks gespeichert wurden. Dieser wird nun benutzt um auf doWork() Parallel auf allen benötigten Nodes zuzugreifen. Die Nodes erhalten nun ihren jeweiligen Teil der Arbeit. Sobald sie diesen bearbeitet und zurückgeschickt haben, werden die Teile zusammengesetzt und an den Client zurückgesendet.

Vorstellung Primfaktorzerlegung

Das Plugin Primfaktorzerlegung ist im Projekt Primes verordnet und implementiert das Interface IPlugin.

Grundsätzlich basiert die Berechnung mithilfe des Plugins darauf, dass das Intervall in dem alle Primfaktoren berechnet werden(z.B 2-100) auf eine Anzahl Nodes verteilt wird um die Berechnung zu beschleunigen (z.B Node1 -> 2-50 Node2-> 51-100). Die Last wird gleichmäßig verteilt also das Intervall durch die Anzahl der Nodes geteilt. Das macht die Methode nodeAmount, die einen String mit der Zahl erhält bis zu der die Primfaktoren bestimmt werden sollen und diese dann durch die Anzahl der Nodes die zur Berechnung verwendet werden sollen teilt. Hier wird festgelegt wie viele Nodes ein Plugin bearbeiten.

Die Methode DisplayPlugins() gibt den Namen tatsächlichen Namen des Plugins zurück, nicht den systeminternen Namen.

Die Methode DoOperation() erhält als Parameter einen String der den Bereich angibt in dem die Primfaktoren berechnet werden sollen und gibt einen String mit den Primfaktoren zurück. Die untere Grenze wird darauf überprüft ob sie mindestens 2 ist und anschließend werden die Primfaktoren der Zahlen im der Node zugewiesenen Intervall berechnet.

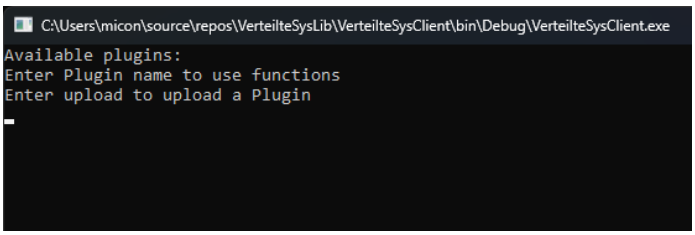
Um das Plugin zu benutzen, muss es, wenn es nicht bereits geladen ist, vom Client hochgeladen werden. Hierzu muss zuerst upload in die CLI vom Client geschrieben werden, dann mit Enter bestätigen woraufhin nach dem Pfad zum Plugin gefragt wird. Nun gibt man den Pfad zur Primes.dll an und bestätigt mit Enter. Nun sollte Primfaktorzerlegung als available angezeigt werden. Um das Plugin nun zu nutzen wird *Primfaktorzerlegung* , Leerzeichen gefolgt von der Zahl bis zu der man die Primfaktoren ausgegeben haben möchte in die CLI eingegeben und mit Enter bestätigt.

Vorstellung PierpontPlugin

Zur Vorstellung des Pierpont Plugins wird die Benutzung beispielhaft dargestellt.

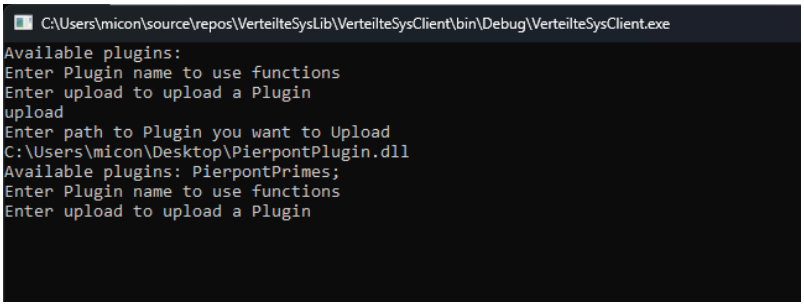
Die Pierpont Primzahlen sind Zahlen die sowohl die Form $1+2^{x}*3^{y}$ haben, wie auch Primzahlen sind.

Um mit Hilfe des Plugins die Pierpont Primzahlen bis zu einer Zahl zu berechnen muss das Plugin zuerst hochgeladen werden.

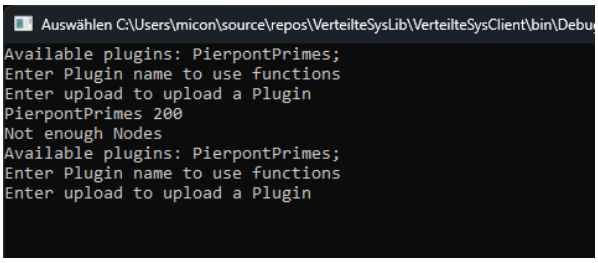


Nachdem man upload gefolgt vom Pfad zur PierpontPlugin.dll durchgeführt hat, wird die Dll vom Client in ein ByteArray eingelesen und an den Server über den PluginService an die UploadFile Methode gesendet. Hier wird das ByteArray des Plugins an den ServiceManager weitergereicht, wo es dann zurück in eine dll konvertiert wird, in eine eigene Directory gespeichert wird und anschließend mit dem MEF recomposed und damit zur Runtime in den Server eingefügt wird. Ebenso wird das byteArray an alle registrierten Nodes geschickt, wo es ebenso wie im Server verarbeitet wird.

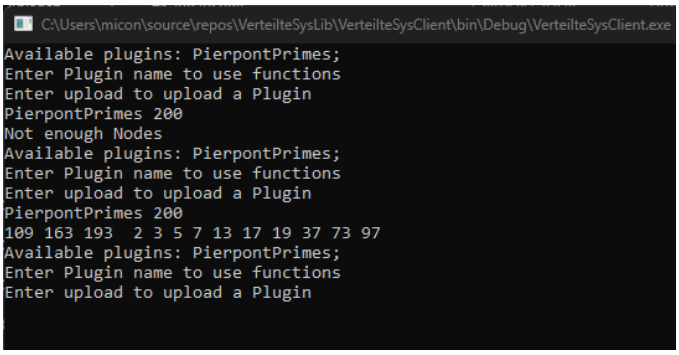
Nun wird DisplayAllPlugins() aufgerufen, was im Server dafür sorgt dass die gesamte Liste der Plugins ihre Namen aus der Funktion DisplayAllPlugins() ausgeben. So werden alle verfügbaren Plugins dem Client angezeigt.



Jetzt kann man mit der Eingabe *PierpontPrimes 200* das Plugin dazu verwenden alle Pierpont Primzahlen bis 200 auszugeben.



Wenn es der Server jedoch gerade nicht genug Knoten zur Berechnung zur Verfügung hat, erhält man diese Ausgabe.



Hat der Server genug Knoten zur Verfügung (2 Konfiguriert für Pierpont) erhält man diese Ausgabe. Das geschieht indem der Client usePlugin mit dem eingegeben Befehl über den PluginService an den Server schickt. Hier wird diese Anfrage vom PluginService welcher das Interface IPluginService implementiert an den ServiceManager weitergeleitet. Dieser teilt nun den erhaltenen Befehl in Parameter und Plugin Name auf, schaut ob es ein Plugin mit diesem Namen gibt und ermittelt mithilfe der nodeAmount Methode des Plugin Interfaces, in einem intArray wie viele Nodes benötigt werden und wie die Arbeit aufgeteilt wird.

Nun wird die Liste _availableNodes abgefragt ob genug Knoten für die Berechnung verfügbar sind und wenn ja werden diese in eine Liste von Knoten die in Benutzung sind überführt und aus der Liste der verfügbaren gelöscht.

Jetzt wird eine Threadsichere Collection outputs generiert in die die Outputstrings der einzelnen Knoten geladen werden können.

Anschließend werden die Befehle die die Knoten benötigen um ihren Berechnungsbereich und das Plugin zu kennen.

Nun werden die Nodes aus der in Benutzung Liste parallel mit ihren Befehlen über die Methode doWork() aktiviert. Nun empfangen die Knoten den Befehl über den PluginService, suchen das richtige Plugin aus der internen Liste und führen die Methode DoOperation() des IPlugin Interface aus.

Im Pierpont Plugin werden nun die untere und obere Grenze aus dem Befehl ermittelt. Zuerst werden alle Primzahlen in diesem Bereich ermittelt und dann werden alle Pierpont Zahlen, also Zahlen die die Form $1+2^{x}*3^{y}$ haben ermittelt und mit der Liste aller Primzahlen verglichen. Die Schnittmenge dieser Listen wird zu einem String gemacht und zurück an den ServiceManager gegeben. Hier werden die Ausgaben aller Knoten zusammengefasst und dann als String zurück an den Client gegeben. Das ist die Ausgabe, die man sieht.

Logging

Das Logging in diesem Projekt ist mit log4net realisiert. Es gibt sowohl den Konsolen appender wie auch den file Appender der ein log.txt in c:/logs/verteilteSysLib speichert.

Es wird die Stufe log.Info, für Informationen verwendet die für einen Benutzer relevant sein könnten.

Es wird die Stufe log.Debug für Informationen während der Laufzeit welche zum Debuggen wichtig sein könnten verwendet.

Es wird die Stufe log.Error für Errors und Exceptions verwendet.