Prototyp für ein mobiles Agentensystem in LVOOP

Frederik Berck und Dr. Holger Brand
GSI Helmholtzzentrum für Schwerionenforschung GmbH, Darmstadt

Kurzfassung

Ein mobiler Agent [1] ist eine Software, die eigenständig eine ihr zugewiesene Aufgabe ausführt, mit ihrer Umgebung interagieren kann und zusätzlich in der Lage ist, mitsamt ihrer Daten zwischen verschiedenen Hosts zu migrieren. Im Gegensatz zu konventionellen objektorientierten Programmiersprachen wie C++ oder Java werden Objekte in LabVIEW nicht als Entitäten behandelt, sondern als passive Datenobjekte, die dem Datenflusskonzept folgen. Die HGF Basisklassenbibliothek [2] stellt LVOOP Klassen bereit, die wichtige Entwurfsmuster datenflusskonform implementieren und die Behandlung von LabVIEW-Objekten als Entitäten ermöglicht. Dabei wird streng zwischen Klassen unterschieden, die passive Datenobjekte (*HGF_Visitable*) repräsentieren und solchen, die jene aktivieren (*HGF_ThreadPool*). Als Beispiel wurde ein stationärer Geräteagent implementiert, der die Machbarkeit demonstriert. In diesem Beitrag wird beschrieben, wie sich dieses Konzept auf mobile Agenten erweitern lässt.

Abstract

A mobile agent [1] is a piece of software which can perform tasks autonomously, interacts with its environment and can move between hosts. In contrast to conventional object oriented languages like C++ or Java, objects are not maintained as entities but as passive data that follow the dataflow paradigm of LabVIEW. The HGF base class library [2] provides LVOOP classes that implement important design patterns with respect to dataflow and enables the treatment of LabVIEW objects as entities. Classes with passive data only (*HGF_Visitable*) are strictly distinguished from classes activating them (*HGF_ThreadPool*). A stationary device agent was implemented as an example to demonstrate the feasibility. This article describes the extension of this concept towards mobile agents.

Die HGF Basisklassenbibliothek

Die HGF Basisklassenbibliothek implementiert diverse Entwurfsmuster, die das datenflusskonforme Behandeln von LVOOP Objekten als Entitäten

erlauben. Unter Anderem werden das Fabrikmuster, der ThreadPool und das Besuchermuster, sowie ein objektorientierter Ereignismechanismus bereitgestellt. Die Fabrik ermöglicht das programmatische Erzeugen initialisierter Objekte. In dem *createObjekt.vi* wird ein Standardobjekt der gewünschten Klasse erzeugt und anschließend durch das *dynamic dispatch initialize.vi* initialisiert. Jede Kindklasse der *HGF_Factory* kann dieses überschreiben, um die eigenen Attribute, die als *Variant*-Attribute übergeben werden, zu initialisieren. Der ThreadPool startet mittels VI-Server Methoden Threads (*HGF_Worker*). Diese Worker aktivieren *HGF_Tasks*, in dem sie das *dynamic dispatch action.vi* des Tasks aufrufen. Mit Hilfe des Ereignismechanismus (*HGF_Event*) können Prozesse synchronisiert und Objekte über Prozessgrenzen hinweg übertragen werden. Das Besuchermuster wird benutzt, um Objekte zu manipulieren, deren Objektdraht nicht direkt zugänglich ist. Das *HGF_Visitable* akzeptiert (*accept.vi*) einen *HGF_Visitor* und ruft dessen *visit.vi* mit sich selbst als Parameter auf. *HGF_Visitor* Kindklassen rufen in den zugehörigen *override* VIs die öffentlichen Methoden der *HGF_Visitable* Kindklassen auf.

Agentenbasis- und Hilfsklassen

Im Rahmen einer Diplomarbeit [4] wurde ein Prototyp für ein **mo**biles **Ag**enten**sy**stem erstellt, das die HGF Basisklassenbibliothek erweitert. Die Basisklasse für mobile Agenten ist *MoAgSy_Agent* und spezialisiert *HGF_Visitable*. Sie akzeptiert Kindklassen von *MoAgSy_Visitor*, welcher eine Spezialisierung von *HGF_Visitor* ist. Bei diesen Agenten handelt es sich um passive Datenobjekte, die mit Hilfe des *ThreadPools* und eines speziellen Tasks, *MoAgSy_Engine*, aktiviert werden. *MoAgSy_Visitor* Objekte bestimmen, wann sie aktiv werden und welche öffentlichen Methoden ausgeführt werden sollen.
Die *HGF_GPG* Klasse nutzt Kommandozeilenfunktionen von GnuPG [3] um in LabVIEW programmatisch die Verschlüsselungs- und Signaturfunktionen von GnuPG verfügbar zu machen. Die *HGF_SV* Bibliothek ergänzt das Basisklassensystem um Klassen, die einen Ereignismechanismus für Umgebungsvariablen, *Shared Variables*, beinhalten. LabVIEW alleine ermöglicht das programmatische Erzeugen bzw. Zerstören von Umgebungsvariablen nicht. Diese Funktionalität wird durch das *Data Logging and Supervisory Control Module* (DSC) ergänzt und durch die *HGF_DSC* Klassenbibliothek integriert.

Der Prototyp für das mobile Agentensystem

Als lokale Schnittstelle für Agenten dient eine Host-Applikation. Diese besteht aus einer Host-Klasse, die in ihren Attributen die nötigen Daten für die Verwaltung bereitstellt und einem *runHost.vi*, welches sowohl das GUI, als auch die nötige Infrastruktur zum Empfangen und Aktivieren von Agenten bereitstellt. Für die Kommunikation werden in diesem Prototyp

Umgebungsvariablen verwendet. Ein *ThreadPool* startet zwei *Worker*. Diese aktivieren jeweils einen Task, der einen *HGF_DSCManager* bzw. einen *HGF_DSCMonitor* kapselt. Diese spezialisieren die *HGF_Visitable* Klasse. Die Tasks warten in Schleifen mit Hilfe des Ereignismechanismus auf geeignete Besucher, die notwendige *Shared Variables* beim Manager erzeugen und beim Monitor registrieren lassen. Der Monitor reagiert auf Änderungen von Umgebungsvariablen und sendet die zugehörigen Besucher, die den neuen Wert in ihren Attributen transportieren, in diesem Fall eine im folgenden erläuterte Nachricht, an den Host oder die adressierten Agenten.
Als Nachrichtencontainer für die Kommunikation innerhalb des Agentensystems dient eine Message-Klasse (*MoAgSy_Message*), deren Objekte in ihren Attributen sowohl Textnachrichten als auch Besucher- oder Agentenobjekte enthalten können. Die Message-Objekte werden mittels *flatten to string.vi* in eine Zeichenkette gewandelt, welcher anschließend an eine Shared Variable übertragen wird. Das bei dem Monitor registrierte Ereignis liefert den String zurück. Anschließend kann mittels *unflatten from string.vi* das Message-Objekt rekonstruiert werden. Da private Attribute durch Nutzung der *flatten to-* bzw. *unflatten from string* VIs in Verbindung mit dem öffentlichen Transportkanal der Umgebungsvariablen für jeden sichtbar werden, existiert zusätzlich die Möglichkeit die Zeichenkette durch Nutzung der Methoden der HGF_GPG Klasse asymmetrisch zu verschlüsseln und somit nur dem gewünschten Ziel die Entschlüsselung zu ermöglichen.
Die Übertragung von Agenten zwischen Hosts und Klientenapplikationen erfolgt durch den Austausch mehrerer Nachrichten. Zuerst wird anhand einer Positivliste auf dem Zielhost die Berechtigung überprüft, den Agenten zu aktivieren. Falls dies erlaubt ist, wird anschließend versucht, die zugehörigen Klassen aus einem lokalen Verzeichnis in den Speicher zu laden. Durch diesen Mechanismus können auch im Laufzeitsystem zuvor unbekannte Agentenklassen geladen werden. Bei Erfolg wird der Agent zum Abschluß verschlüsselt übertragen und aktiviert.
Die Aktivierung eines Agenten erfolgt, indem der Host mittels *MoAgSy_Factory* einen neuen Engine-Task (Bild1) erzeugt, der den Agenten als Initialisierungsparameter erhält, und an einen ThreadPool übergibt. Der Engine-Task treibt eine synchrone Zustandsmaschine (Bild2), welche mit Hilfe des Statechart Module implementiert ist. In der initialen Transition wird der Agent in die internen Attribute der Zustandsmaschine übernommen. Die Zustandsmachine initialisiert die Kommunikations-schnittstelle des Agenten, in dem eine neue Umgebungsvariable erzeugt und ein Ereignis für diese Variable registriert wird. Anschließend wechselt die Zustandsmachine in den Zustand ***Waiting*** und die Engine wartet auf eingehende Message-Objekte. Eingehende Nachrichten bewirken einen Zustandswechsel. Zuerst wird Empfänger und Art der Nachricht überprüft. Textnachrichten und Agenten haben zur Folge, das jeweils ein *dynamic dispatch* VI aufgerufen wird, in dem

spezialisierte Agenten-Klassen Reaktionen auf die jeweilige Nachricht implementieren. Besucher werden im Zustand ***Visitor*** (Bild3) auf den Agenten angewendet. Die Übertragung eines Reisebesuchers, *MoAgSy_Visitor_travel*, löst die Migration zu einem anderen Host aus.
Um nicht nur reaktive, sondern auch autonom agierende Agenten zu ermöglichen, verfügt die Agenten-Basisklasse über einen Timeout-Besucher. Dieser wird nach Ablauf einer bestimmten Zeitspanne vom Ereignismechanismus des Engine-Tasks an die Zustandsmachine übergeben und kann somit eine wiederkehrende Aktion des Agenten triggern. Darüber hinaus kann der Agent auch sich selbst asynchrone Nachrichten senden.
Durch implementieren spezialisierter Agenten und zugehöriger Besucher kann das mobile Agentensystem jederzeit erweitert werden, wodurch anwendungsspezifische Applikationen erstellt werden können.
Ein vom Hostsystem unabhängiges VI wird benötigt, um mindestens einen ersten Agenten zu initialisieren und an einen Host zu senden. Während die Initialisierung für jede Agenten-Klasse individuell implementiert werden kann, kann die Übertragung an den Host mittels *startAgent.vi* erfolgen, welches von der Fabrik des mobilen Agentensystems bereitgestellt wird.

Status und Ausblick

Mit Hilfe einfacher Agenten wurde die Funktionalität der Basisklassen und des Hostsystems demonstriert und gezeigt, dass die HGF Basisklassen eine tragfähige Grundlage für skalierbare, ereignisgesteuerte und verteilte objektorientierte Projekte in LabVIEW bilden. Im nächsten Schritt muss die Performanz und Stabilität des mobilen Agentensystems mit Hilfe einer größeren Applikation grundlegend getestet und überprüft werden. Zusätzlich müssen weitere Hilfsmittel entwickelt werden, um die Implementierung von Agenten-Bibliotheken bzw. die Problemdiagnose im Fehlerfall zu vereinfachen und die Verwaltung von sowohl aktiven als auch persistenten Agenten zu ermöglichen.

Literatur

[1]D.Lange M.Oshima "Programming And Deploying Java Mobile Agents With Aglets", Addison Wesley, ISBN 0-201-32582-9
[2]H.Brand, D.Beck, F.Berck "Machbarkeitsstudie: LVOOP basiertes Agenten-System", erschienen in "Virtuelle Instrumente in der Praxis - Begleitband zum Kongress VIP 2009", Rahman Jamal, Hans Jascchinski (Hrsg.), Hüthig Verlag, ISBN 978-3-7785-4057-2
[3]http://www.gnupg.org/
[4]F.Berck "Prototyp für ein mobiles Agentensystem in NI LabVIEW", Hochschule Darmstadt University of Applied Science, FB EIT, 2010, http://wiki.gsi.de/pub/NIUser/LVMobileAgentSystem/DiplomarbeitFrederikBerck.pdf

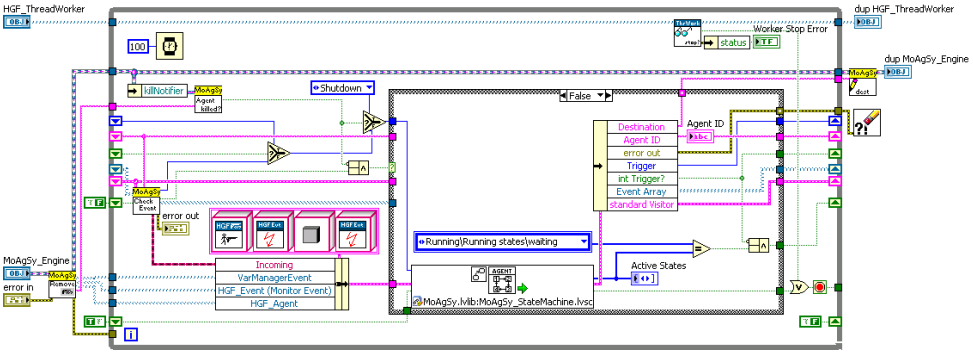

Bild1: Dynamic dispatch action.vi des Engine-Tasks

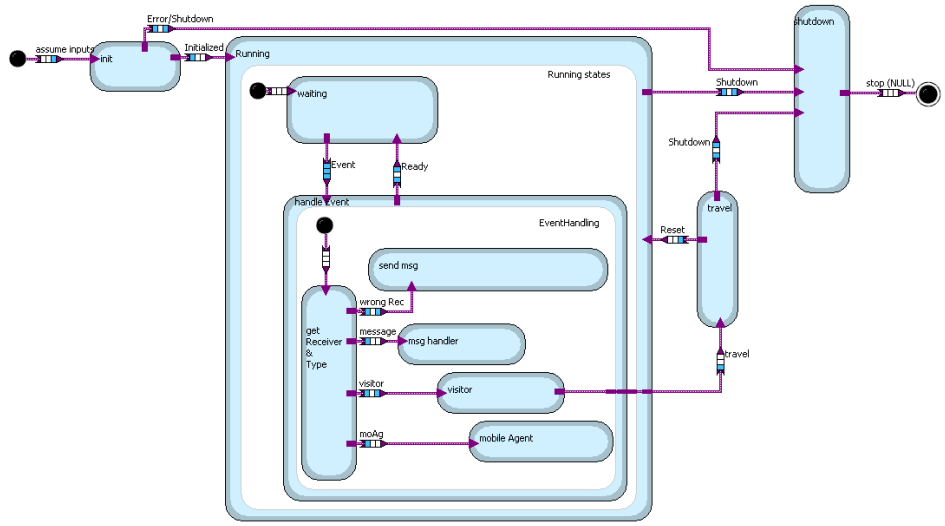

Bild2: Zustandsdiagramm der Agenten-Zustandsmachine

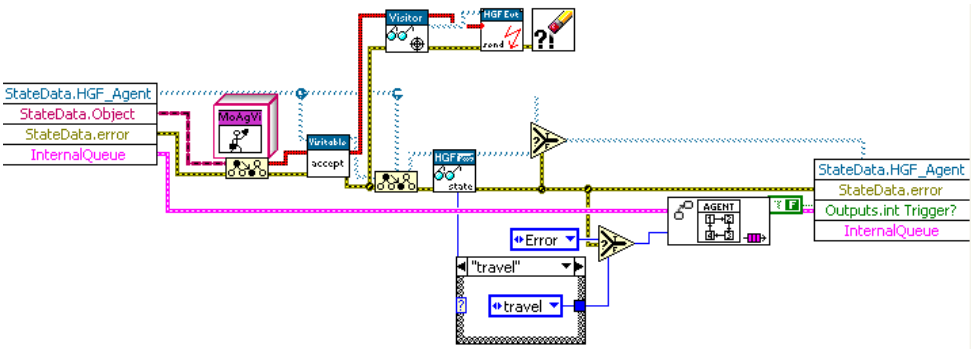

*Bild3: Eingangsaktion des Zustands **Visitor** der Agenten-Zustandsmachine*