Institut für Systemsoftware

O.Univ.-Prof. Dr. Hanspeter Mössenböck

JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

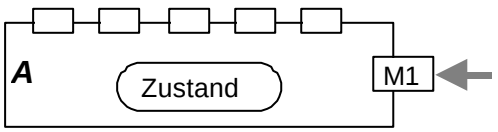
Eine Kommando-Oberfläche für .NET

In .NET (wie auch in vielen anderen Systemen) haben Programme nur einen einzigen Eintrittspunkt (ihre Main-Methode), d.h. das Programm wird immer über diesen Eintrittspunkt gestartet. Üblicherweise bietet das Programm dann Menüs an, um unterschiedliche Aufgaben auszuführen. Es ist aber selten möglich, zu einem Programm neue Funktionalität hinzuzufügen, die zum Zeitpunkt der Erstellung des Programms noch nicht bekannt war. Dazu fehlen die Menüpunkte, und es gibt meist auch keinen Mechanismus, um diese neue (statisch noch unbekannte) Funktionalität nachzuladen.

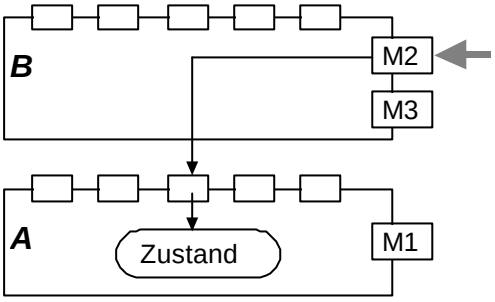
Das Oberon-System der ETH Zürich bietet hier eine interessante Lösung an: Programme können dort mehrere Eintrittspunkte haben. Jede statische parameterlose Methode (genannt *Kommando*) kann vom Benutzer direkt gestartet werden, indem er ihren Namen in ein Textfenster schreibt und ihn anklickt (meist steht der Name bereits in einem Textfenster und muß nur noch angeklickt werden). Da ein Programm mehrere solche Methoden haben kann, hat es mehrere Eintrittspunkte. Dadurch kann man sich viele Menüs sparen, da die verschiedenen Aufgaben direkt durch Kommandos gestartet werden können.

Beim Aufruf eines Kommandos wird die Klasse, zu der die Kommandomethode gehört, geladen und die Methode wird ausgeführt. Anschließend bleibt die Klasse geladen und behält ihren Zustand bei (d.h. statische Variablen behalten ihre Werte). Wenn ein weiteres Kommando dieser Klasse aufgerufen wird, muß die Klasse nicht mehr geladen werden, sondern die Kommandomethode wird sofort gestartet und kann auf den vorhandenen Zustand der Klasse zugreifen. Auf diese Weise können mehrere Kommandos einer Klasse über den gespeicherten Zustand der Klasse zusammenarbeiten.

Kommandos erlauben es auch, ein Programm um statisch unbekannte Funktionalität zu erweitern. Angenommen, das Hauptprogramm (Klasse *A*) wurde durch ein Kommando (Methode *M1*) geladen:



Nun wird ein weiteres Kommando *B.M2* (d.h. Methode *M2* der Klasse *B*) aufgerufen. Dadurch wird *B* geladen und *M2* gestartet. *M2* kann nun auf die public-Schnittstelle von *A* zugreifen, Methoden von *A* aufrufen und somit Operationen auf die Daten von *A* ausführen.



A wurde dadurch um neue Funktionalität (*B*) erweitert, die zum Zeitpunkt als *A* implementiert wurde noch unbekannt war. Wird nun auch *B.M3* aufgerufen, braucht *B* nicht mehr geladen zu werden, sondern *M3* wird sofort gestartet und kann z.B. wieder auf *A* zugreifen.

Aber wie kann *A* die Funktionalität von *B* nutzen, wo es doch keine Menüeinträge hat, um diese Funktionalität aufzurufen? Die Lösung ist einfach: *B* installiert in einem Fenster von *A* neue Menüs, die bei ihrer Auswahl Methoden von *B* aufrufen. Auch das ist ein Beispiel, wie ein Programm um neue (statisch unbekannte) Funktionalität erweitert werden kann.

Das Ziel ist hier ähnlich wie bei Eclipse: neue Funktionalität soll dynamisch hinzugeladen werden. Allerdings müssen bei diesem Modell die "Plugins" nicht durch XML beschrieben werden, sondern jede beliebige Klasse kann als Plugin fungieren. Auch Extension Points müssen nicht durch XML beschrieben werden. Der Extension Point einer Klasse ist einfach ihre public-Schnittstelle, die von allen Plugins verwendet werden kann.

Ihre Aufgabe in diesem Projekt ist es, unter .NET ähnliche Mechanismen wie in Oberon zu implementieren. Es soll dadurch möglich sein, Programme mit mehreren Eintrittspunkten zu realisieren. Neue Funktionalität soll über Kommandos nachgeladen werden können. Eine weitere Aufgabe des Projekts ist es, diese Mechanismen in zwei Prototyp-Anwendungen auszuprobieren.

Kommandos über Reflection aufrufen

- Ein Kommando ist eine statische Methode (z.B. *Clipboard.Copy*), deren Namen man in ein beliebigen Textfenster eingeben kann. Wenn man mit der mittleren Maustaste auf den Namen klickt, wird die Klasse mittels Reflection geladen (falls sie nicht bereits geladen ist) und die Methode wird aufgerufen. Nach Beendigung der Methode bleibt die Klasse geladen, wie das in .NET üblich ist.
- Zur Vereifachung des Aufrufs ist eine Hilfsmethode zu schreiben, mit der man ein Kommando z.B. durch *Call("Clipboard.Copy")* aufrufen kann.
- Somit kann man Texte erstellen und abspeichern, die eine Menge von Kommandos enthalten. Man braucht diese Texte dann nur in einem Fenster zu öffnen und das gewünschte Kommando anklicken. Die gespeicherten Texte fungieren sozusagen als textuelle Menüs.
- Eventuell sollte man verlangen, daß jene Methoden, die als Kommandos aufrufbar sind, mit einem [Command]-Attribut gekennzeichnet werden müssen.

Abbildung von Klassen auf Assemblies (DLLs)

- Da man in .NET aus dem Namen einer Klasse nicht auf den Namen des Assemblys schließen kann, das diese Klasse enthält, muß es eine Abbildung von Klassennamen auf Assemblynamen geben. Vorschlag: beim Start der Kommando-Oberfläche werden alle DLLs im aktuellen Verzeichnis gelesen und es wird eine Hashtabelle aufgebaut, die einen Klassennamen auf einen Assemblynamen abbildet.
- Über Programmparameter kann man eventuell noch weitere Verzeichnisse angeben, in denn nach DLLs gesucht werden soll.
- Da verschiedene Assemblys gleichnamige Klassen enthalten können, kann man in einer verfeinerten Version vorsehen, daß ein Kommandoname auch den Assemblynamen enthält, z.B.: *[MyAssembly]Clipboad.Copy*
- Eventuell sollte man in die Hashabelle nur solche Klassen aufnehmen, die zumindest eine statische Methode enthalten, denn nur diese können als Kommandos aufgerufen werden.

Fenstersystem

- Die Hauptapplikation sollte ein Kommandofenster öffnen, das bereits einen Text mit Standardkommandos enthält, die man anklicken kann. Durch ein bestimmtes Kommando kann man weitere Kommandofenster mit anderen Kommandosammlungen öffnen. Die Texte in den Kommandofenstern sollen auch abgespeichert werden können.
- Programme sollen Text in ein spezielles Kommandofenster ausgeben können. Insbesondere sollen dadurch Kommandonamen ausgegeben werden können, die der Benutzer dann wieder anklicken kann.
- Die Applikation soll eine Liste aller Kommandofenster verwalten. Es soll eine Funktion geben, die in all diesen Fenstern die aktuelle Selektion bzw. das Caret sucht. Der Text in diesem Fenster und die Position der Selektion bzw. des Carets sollen als Parameter für andere Kommandos verwendet werden können.
- Es soll einen Reader geben, den man an eine bestimmte Position eines Textes in einem Kommandofenster setzen kann. Mit Hilfe dieses Readers sollen Kommandoparameter von dieser Stelle gelesen werden können. Beim Aufruf eines Kommandos soll der Reader implizit auf die Textposition nach dem Kommandonamen gesetzt werden. Üblicherweise stehen dort die Kommandoparameter. Wenn dort aber das Zeichen ^ steht, soll der Reader automatisch auf die Position des aktuellen Carets bzw. der aktuellen Selektion gesetzt werden.
- Es sollte eine statische Variable *cmdWindow* geben, die (falls nicht null) dasjenige Fenster angibt, aus dem ein Kommando aufgerufen wurde.

Mögliche Erweiterungen

- Es soll einen Zustand geben, der als Menge von Name/Wert-Paaren implementiert ist. Beliebige Werte sollen dort unter beliebigen Namen abgelegt werden können und zwar sowohl durch Kommandos als auch von einem Programm aus. Es soll möglich sein, sich alle belegten Namen mit ihren Werten anzusehen.
- Eventuell ist zu überlegen, Kommandos auch mit Parametern zuzulassen. Dabei reicht es wahrscheinlich, konstante Parameter der Typen *int*, *float*, *string* zuzulassen. Eventuell könnte man auch benannte Parameter zulassen, die dann Werte aus der oben beschriebenen Zustandsmenge bezeichnen und denen formale Parameter vom Typ *Object* gegenüberstehen.

- Eventuell könnte ein Kommando auch eine Funktion sein, die einen Wert vom Typ *Object* liefert, der dann unter einem bestimmten Namen in der Zustandsmenge abgelegt wird.

Hauptprogramm

Ihr Hauptprogramm besteht aus einer Klasse mit folgender Funktionalität:
- Kommandofenster verwalten
  - Öffnen und Schließen solcher Fenster
  - Abspeichern des Textes eines Fensters in eine Datei
  - Liste aller Fenster verwalten sowie Caret und Selektion darin suchen
- Klicks auf Kommandos verarbeiten
- Zuordnung zwischen Klassennamen und DLLs verwalten
- Zustand verwalten

Betreuung

o.Univ.-Prof. Dr. H. Mössenböck
Teamgröße: 2 Personen

Vorschläge für Anwendungen

- Ein einfacher Texteditor, der mittels Kommandos auf folgende Weise erweitert werden kann:
  - Menüpunkte und Menüeinträge hinzufügen
  - neue Panels hinzufügen, z.B. eine Verzeichnissicht
  - neue Funktionalität (z.B. Spell Checking) direkt über Kommandos aufrufen
- Ein Kommandofenster, das neben Texten auch eine Sammlung von Objekten anzeigt. Das Fenster kennt allerdings nur abstrakte Objekte. Daraus abgeleitete konkrete Objekte können mittels Kommandos hinzugefügt werden:
  - Buttons
  - Popup-Menüs
  - eine Uhr