Java Beans Glasgow - Komponentenarchitektur

The Drag and Drop Subsystem for the Java Foundation Classes
+
The JavaBeans Activation Framework

1 Einordnung

Das Drag and Drop Subsystem und das JavaBeans Activation Framework sind neu beim JDK 1.2 und unter dem Codenamen „Glasgow“ zusammen mit noch anderen Erweiterungen veröffentlicht worden. Diese Ausarbeitung soll einen kleinen Überblick über die Neuerungen liefern.

2 The Drag and Drop Subsystem for the Java Foundation Classes

2. 1 Motivation

Das Drag and Drop Subsystem soll es dem Entwickler ermöglichen, ein platformunabhängiges DnD für eine grafische Benutzeroberfläche unter Java zu implementieren. Dabei wurde Wert darauf gelegt, das sowohl reine Java-Anwendungen, als auch plattformabhängige fremde Anwendungen untereinander interagieren können. Dazu ist ein erweiterbares Datentyp-System als eine Erweiterung des existierenden datatransfer-Packages basierend auf dem MIME-Standard integriert. Außerdem ist eine Unterstützung für unterschiedliche Eingabegeräte, welche um beliebige Features erweiterbar ist, vorgesehen.

2. 2 Überblick über die Elemente einer DnD-Operation

DnD ist eine direkte Manipulation bzw. der Austausch von Daten zwischen zwei Entitäten, welche logisch mit den Elementen des GUI verbunden sind. Der Benutzer führt diese mit einem von ihm gewählten Eingabegerät durch. DnD kann dabei ein ständiges Feedback betreffs der DnD-Möglichkeiten während der Navigation über die GUI-Elemente liefern. Eine DnD-Operation kann man dabei in folgende Teilszenarien gliedern (nicht sequentiell):

- Eine *DragSource* entsteht in Zusammenhang mit einem Element des GUI (*Component*) um eine DnD-Operation zu initialisieren

- Eine oder mehrere *DropTargets* entstehen/verfallen im Zusammenhang mit einem Element des GUI, einer DnD-Operation und zu transportierenden Daten.
- Ein *DragGestureRecognizer* steht im Zusammenhang mit einer *DragSource* (und einer *Component*) um eine Drag-Operation zu entdecken und zu verfolgen
- Sollte der *DragGestureRecogmizer* eine solche Operation entdecken, meldet er diese an den zuständigen *DragGestureListener*
- Dieser liefert zusammen mit der *DragSource* ein Feedback für den Benutzer (animierter Cursor und/oder gerendertes Image der Elemente der Operation)
- Sollte der Benutzer innerhalb einer DnD-Aktion über ein Element des GUI (*Component*) gelangen, das als *DropTarget* registriert ist, so erhält die *DragSource* eine Notiz, um ein *DragOver*-Feedback zu erzeugen und das *DropTarget* eine Notiz, um ein *DragUnder*-Feedback zu erzeugen (basierend auf den unterstützten DnD-Operationen und dem jeweiligen Datentyp)
- Dabei manipuliert die *DragSource* den jeweiligen Cursor und das *DropTarget* das verbundene Element des GUI (*Component*)
- Die angebotenen DnD-Operationen, sowie die unterstützten Datentypen werden aus der Schnittmenge der Operationen und Datentypen von *DragSource* und *DropTarget* gebildet
- Sollte der Benutzer die DnD-Operation (erfolgreich) abschließen, so erhalten die *DragSource* und das *DropTarget* eine Information über die Parameter der Operation (Typ der Operation, Datentyp)

2.3 Event-Handling einer DnD-Operation

Nach einer gestarteten DnD-Operation werden über den *DragGestureRecognizer* und die an der Aktion beteiligten *Listener*-Klassen Events ausgetauscht, mit deren Hilfe der Status der DnD-Operation immer eindeutig identifizierbar ist. Dafür können sich *EventListener* mit Hilfe der Methoden *registerListener()* und *unregisterListener()* für eine DnD-Operation registrieren lassen. Sollte eine *DragSource* sich einmal nicht in einem passenden Zustand befinden oder nicht zu dem jeweiligen *DragGestureRecognizer* kompatibel sein, so wird eine entsprechende Exception ausgelöst.
Im folgenden soll nur noch grob auf das oben beschriebene Szenario der DnD-Operation des Benutzers eingegangen werden. So sollen die beiden folgenden Grafiken nur einen kleinen Einblick in die Komplexität der Aktionen liefern. Die erste Grafik beschreibt dabei die Aktionen einer potentiellen *DragSource*, wobei passend zu den Aktionen des Benutzers schon ein Feedback erzeugt werden kann, ob z.B. ein Element des GUI eine DnD-Operation als Quelle unterstützt. Die zweite Grafik stellt dann den Ablauf während einer DnD-Operation dar, wobei die Auswahl eines *DropTarget* verdeutlicht werden soll.

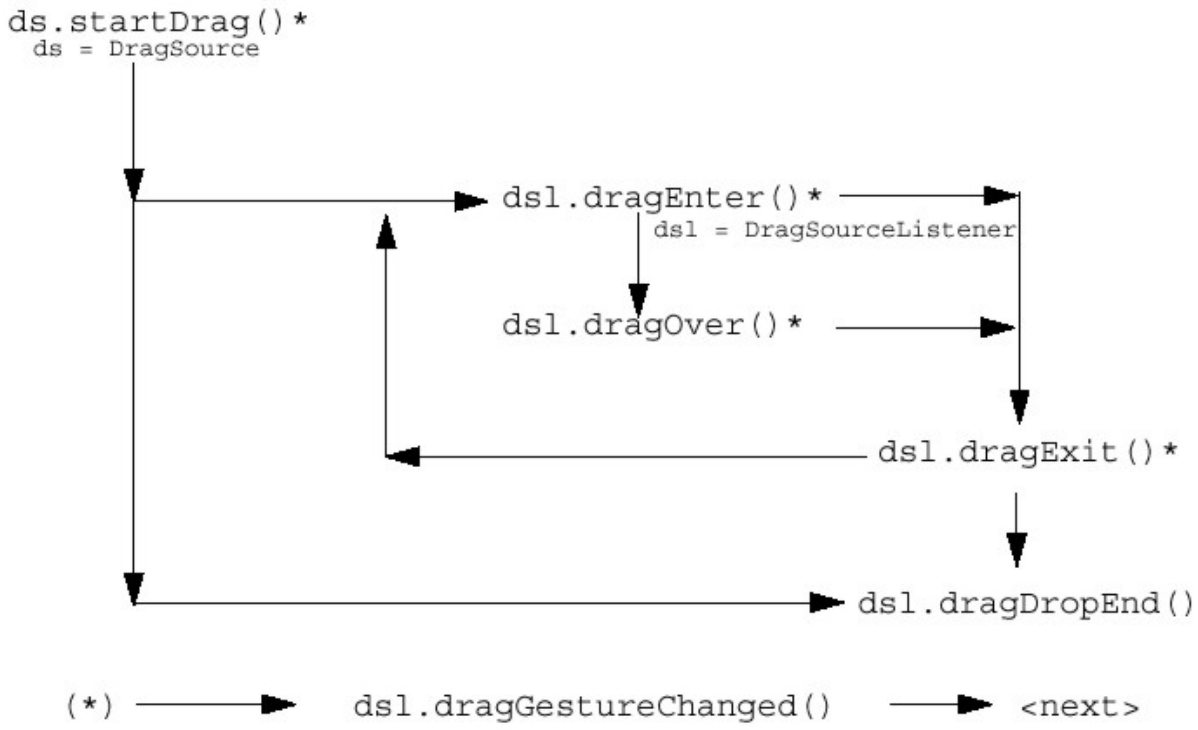

Bild 1: *DragSourceListener* vor einer DnD-Operation

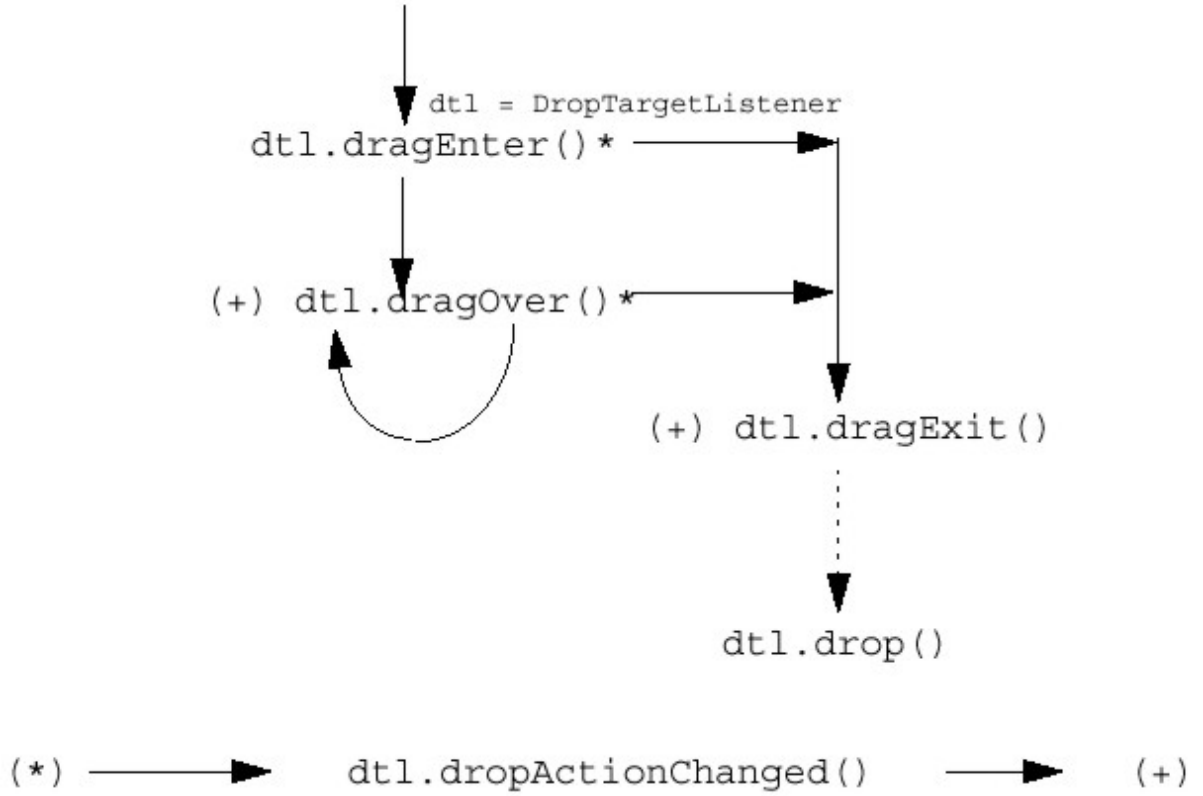

Bild 2: *DropTargetListener* während einer DnD-Operation

2.4 Datentransfer nach der DnD-Operation

Wenn nach der (erfolgreich) abgeschlossenen DnD-Operation des Benutzers Daten zwischen zwei Applikationen ausgetauscht werden sollen, so sollte das zwischen zwei Java-Applikationen kein Problem sein, da das Java-Datentyp-System auf den MIME-Typen basiert, und sich somit die Applikationen nur den jeweiligen MIME-Typ gegenseitig bekanntmachen müssen. Komplizierter wird die Angelegenheit, wenn der Datentransfer über die Grenzen der JavaVirtualMachine hinausgeht. Dies passiert, wenn Daten zwischen einer Java-Anwendung und einer Applikation z.B. des zugrunde liegenden Systems ausgetauscht werden sollen. Diese Anwendung des Systems (Heimat-Anwendung) hat keinerlei Wissen über das Java-Datentyp-System.

- Sollen nun Daten einer Heimat-Anwendung in eine Java-Anwendung transferiert werden, so muß die Quelle für die (De-)Kodierung und das Ziel für den Transport der Daten verantwortlich sein
- Die Daten werden dann gekapselt in einen *InputStream* (*java.io.InputStream* oder eine *Subclass* dessen)
- Das bedeutet, daß jeder Datentyp, der durch eine Klasse repräsentiert ist die *java.io.InputStream* erweitert, über die Grenzen der JVM ausgetauscht werden kann
- Das wiederum bedeutet, daß der Entwickler einen MIME-Typ definieren muß, der die Struktur des Heimat-Datentyps beschreibt (und *java.io.InputStream* erweitert), mit dessen Hilfe man den *InputStream* dekodieren kann

Sind diese Hürden genommen, sollte die DnD-Operation samt Datentransfer erfolgreich abgeschlossen sein und das DnD-System geht wieder in den in Bild 2 geschilderten Zustand über. Eine neue DnD-Operation kann initialisiert werden.

Weitere detailliertere Informationen, insbesondere zur Implementation sind z.B. im „Proposal for a Drag and Drop subsystem for the Java Foundation Classes“ nachzulesen, welches auf im Internet u.a. unter folgender Adresse zu finden sein müsste: http://java.sun.com

3 The JavaBeans Activation Framework (JAF)

3.1 Motivation

Weder JavaBeans noch Java stellen eine einheitliche Strategie zur Verfügung, um Daten zu typisieren, die von einer Software unterstützten Datentypen herauszufinden oder Datentypen mit einer bestimmten Software zu verknüpfen. Das JAF soll nun als Standard-Erweiterung zur Java-Plattform folgende Dienste zur Verfügung stellen:

- Es bestimmte den Typ von beliebigen Daten
- Es kapselt den Zugriff auf Daten
- Es findet die Operationen, die auf einen bestimmten Datentyp möglich sind, heraus
- Es instantiiert eine Softwarekomponente, die mit einer bestimmten Operation eines Datentyps verknüpft ist

3.2 Überblick

Die folgende Grafik soll einen Überblick über die Struktur des Frameworks liefern:

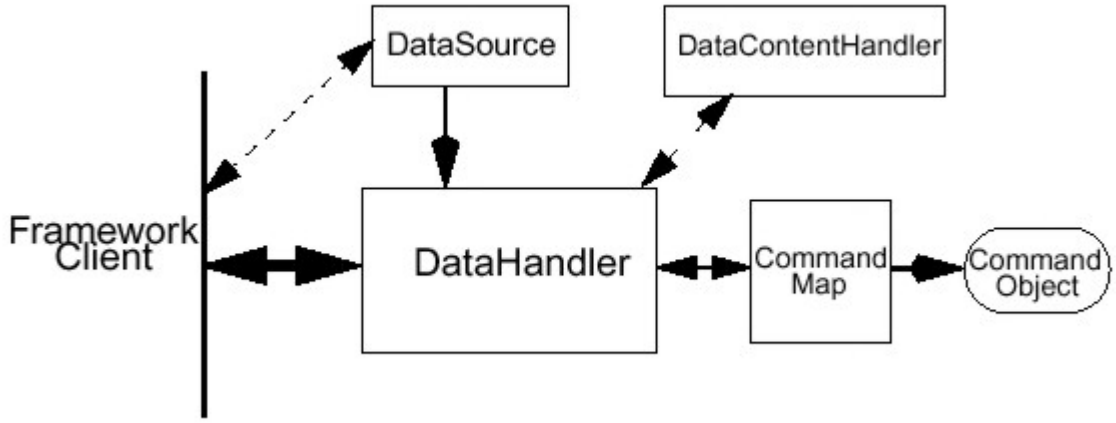

Bild 3: Das JavaBeans Activation Framework (Überblick)

Im Zusammenspiel mit anderen Subsystemen soll die *DataHandler*-Klasse ein einheitliches Interface zur Verfügung stellen. D.h. der normale Zugriff auf das Framework sollte über den *DataHandler* stattfinden, wobei die einzelnen Komponenten auch unabhängig davon „funktionieren“.
Die *DataSource*-Klasse ist ein Interface, daß das Daten enthaltende Objekt kapselt. Es liefert entweder einen Datenstrom oder einen String zurück. Der String definiert dabei den MIME-Typ, der die Daten beschreibt. Es können Klassen für allgemeine Datentypen (Web, File-System, ftp, ...) definiert werden, wobei auch eine Anpassung an Datentypen des Benutzers (spezielle Datentypen)

möglich ist. Ist die *DataSource* einmal im *DataHandler* gesetzt, kann der Client die Operationen, die auf dem Datentyp möglich sind bestimmen. Im JAF sind bereits zwei Klassen vorgesehen: Eine Klasse für den Zugriff auf Dateien und die zweite für den Zugriff auf Daten einer URL.
Das CommandMap-Interface erlaubt es dem Benutzer die Operationen zu erfahren, die auf einen MIME-Typ möglich sind bzw. eine Komponente zu finden, die auf einen MIME-Typ anwendbar ist. Mit Hilfe eines in CommandMap implementierten Mechanismus läßt sich eine Liste erstellen und verwalten, die diese Operationen enthält.
Beans die das CommandObjekt-Interface erweitern können so die JAF-Services nutzen, indem sie direkt auf dessen DataSource und DataHandler zugreifen können. Dies ermöglicht es ihnen den Datentyp herauszufinden und mit den Daten zu arbeiten.

3.3 Beispielszenario File-Viewer

Ein solches Szenario eines File-Viewers (ähnlich Windows-Explorer) läßt sich anschaulich in 3 Teile gliedern:

- Initialisierung
- Erhalten der CommandList (Befehlsliste)
- Ausführen des Befehls

Wir erinnern uns, daß das darunterliegende Daten-Objekt gekapselt wird und der darunterliegende Speichermechanismus abstrahiert wird. Ersichtlich ist ein einheitlicher Datenzugriff.

Initialisierung: Das bei einem File-Viewer zugrunde liegende System ist das File-System. Der Viewer „befragt“ dieses um dessen Inhalte. Der Viewer instantiiert ein DataSource-Objekt für jedes einzelne File im Verzeichnis. Anschließend instantiiert es den DataHandler mit dem DataSource-Objekt als Argument. Das DataHandler-Objekt erlaubt dem Client (File-Viewer) den Zugriff zur CommandMap, welche einen Dienst zur Verfügung stellt, der den Zugriff auf die Befehle erlaubt, die auf den Daten möglich sind.

Erhalten der CommandList: Beim Auswählen eines Files im File-Viewer fragt dieser einer Liste der möglichen Befehle (CommandList) beim DataHandler-Objekt, welches mit dem jeweiligen File verbunden ist. Der DataHandler erhält den MIME-Typ vom DataSource-Objekt und schaut in der CommandMap nach den Operationen, die auf diesem Datentyp möglich sind. Der FileViewer interpretiert diese und gibt dem Benutzer eine Liste zur Auswahl.

Ausführen des Befehls: Nachdem der Benutzer einen Befehl ausgewählt hat benutzt der FielViewer die CommandInfo-Klasse um das korrespondierende Bean zum gewählten Befehl zu erfahren. Anschließend werden die Daten (File) mit Hilfe eines passenden Mechanismus dem Bean assoziiert (DataHandler, Externalization, ...).

Ein weiteres Szenario, sowie Informationen zur Implementation der einzelnen Interfaces und eine Einführung um Beans für das Framework zu schreiben sind in der „JavaBeans Activation Framework Specification“ Version 1.0a zu finden.
http://java.sun.com