

Mathematik Seminar WS 2003: Simulation und Bildanalyse mit Java

Software-Architektur
basierend auf dem
Plug-in-Konzept



Aufteilung:

- Probleme mit „normaler/alter“ Software
- Ziele des Software Engineerings
- Die Plug-in-Architektur als eine Lösung
- Plug-in-basierte Anwendungen
- Plug-in-basierte GUIs
- Die GeoStoch-GUI als Beispiel
- Fazit: Vor/Nachteile der Plug-in-Architektur

Probleme „normale/alte“ Software

- Software wird **immer komplexer**
- Erweiterung durch Dritte nur selten möglich
- **Dritte müssen** sich mit dem ganzen **Programm** auskennen
- Es reicht nicht das Wissen über die einzelnen Schnittstellen und Bibliotheken

Ziele des Software Engineerings

- **Simple und flexibles** Software-Design (KISS)
- Systemerweiterungen benötigen das Spezial-know-how von Dritten deshalb müssen **Erweiterungen von Dritten leicht implementierbar** sein
- Spezielle Unterstützung für **parallele Software-Entwicklung**

Mögliche Lösung durch Plug-ins

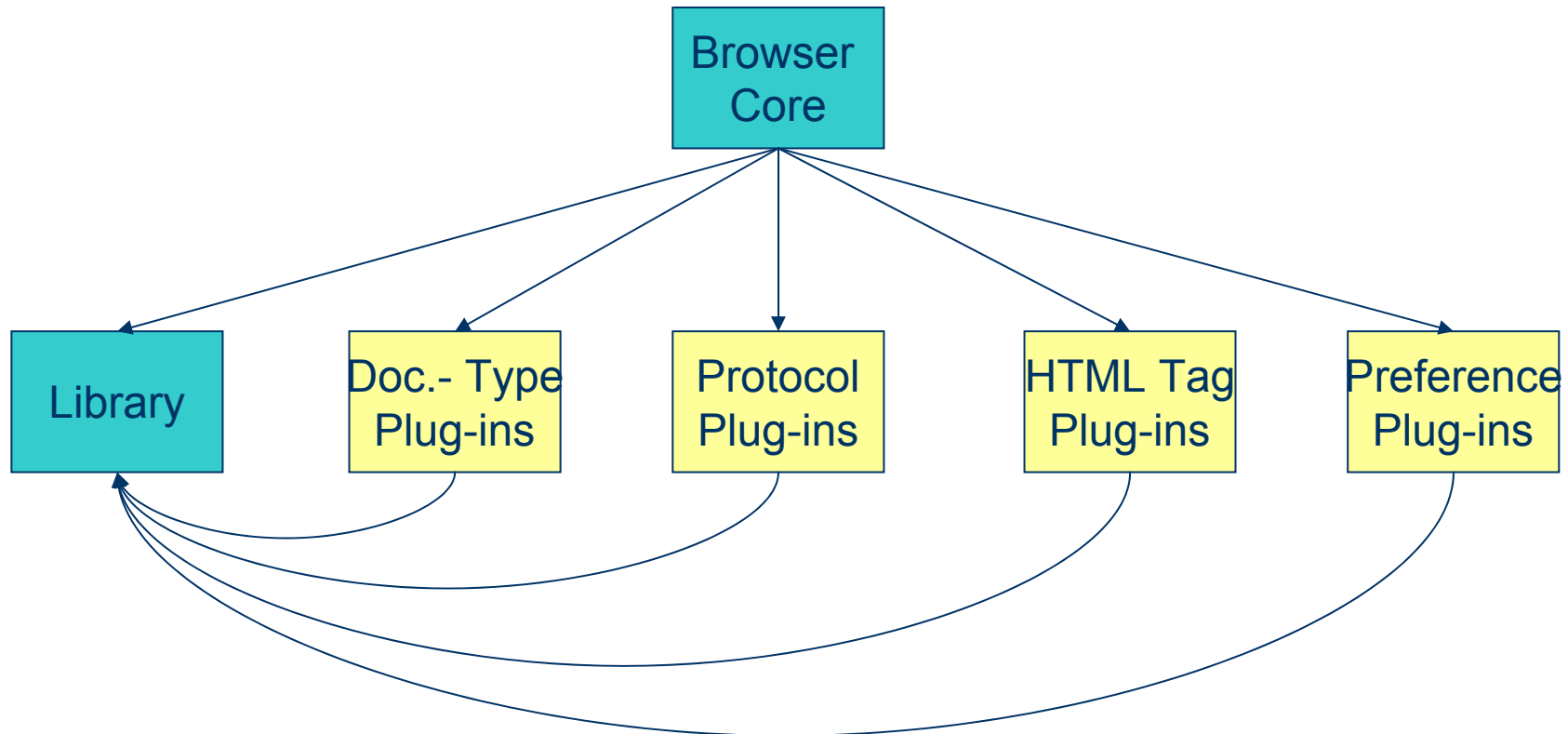
Was ist ein Plug-in?

- Software Komponente
- Nicht alleine lauffähig
- Zur **Laufzeit** nachgeladen (falls gebraucht)
- Implementiert **zusätzliche** oder sogar **wesentliche Funktionalität**
- Kommunikation via **Schnittstelle und Protokolle (Plug-in-Typ)**

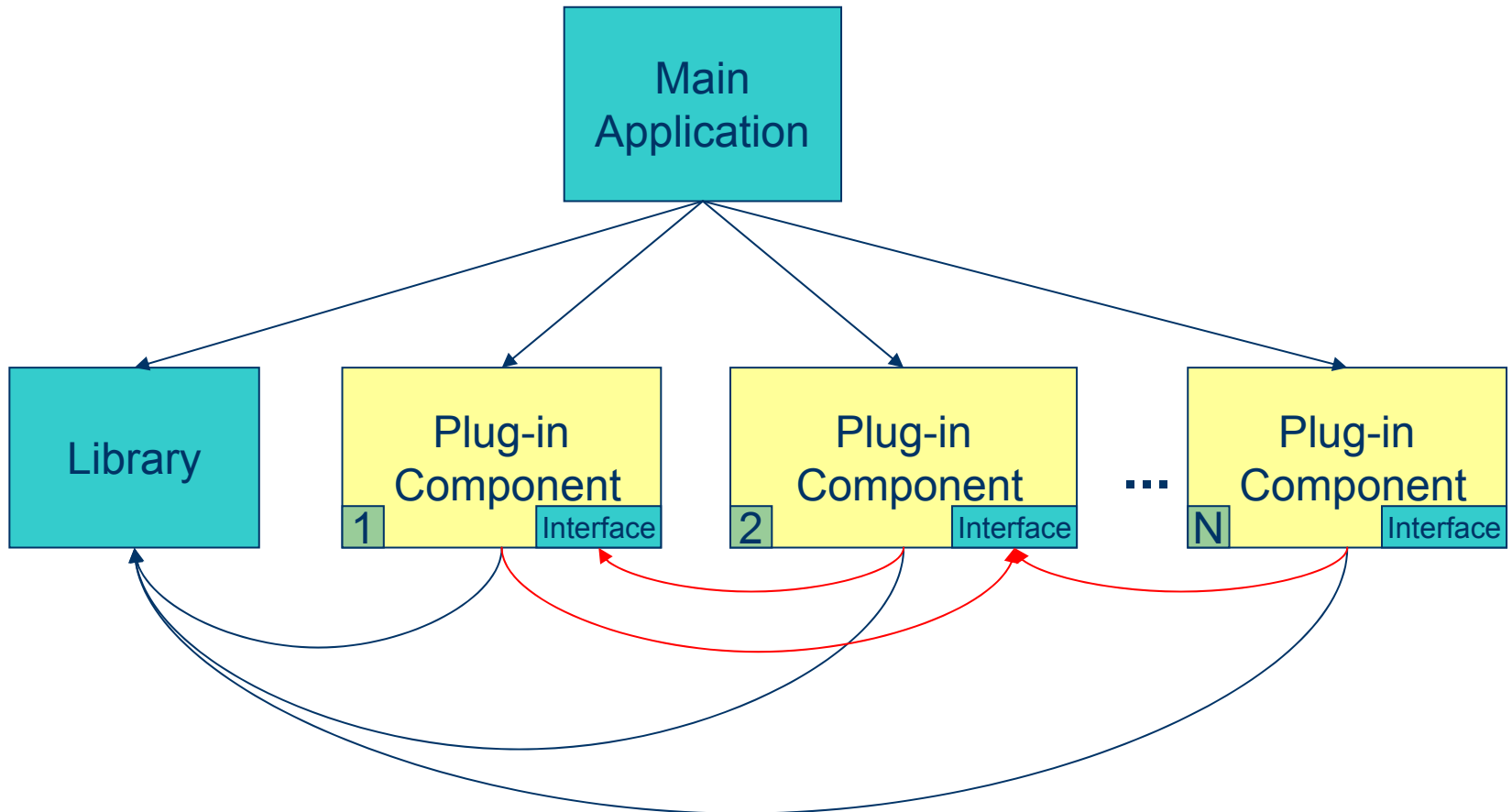
Plug-in-Architektur von einem Browser

- Die **populärsten Anwendungen**, die sich Plug-ins zunutze machen, sind **Internet-Browser** wie Netscape und Internet Explorer.
- **Webbrowser erkennen** während des Laden einer Webseite, **dass ein Plug-in benötigt wird**.
- Das Plug-in wird dann gegebenenfalls installiert, und der **Browser übergibt** die weitere **Verarbeitung und Kontrolle an das Plug-in**.
- Das geschieht für den Benutzer **oft unsichtbar** und **ohne** dass **die Applikation** dafür **verlassen** werden müsste.

Plug-in-Architektur von einem Browser



Basic-Plug-in-Architektur



Beteiligte der Plug-in-Architektur (1)

Haupt-Anwendung (Browser Core):

- führt die **Basis Funktionalitäten** der Anwendung aus, die nicht durch Plug-ins realisierbar sind
- Ist mit der **Bibliothek** ganz oben verbunden und hat **Zugriff zu allen Plug-ins** durch den **Plug-in-loader** und **kommuniziert** mit den Plug-ins über die **Plug-in-interfaces**

Beteiligte der Plug-in-Architektur (2)

Bibliothek (Library):

- beinhaltet **generelle Routinen**, die sonst bei mehreren Teilen separat implementiert werden müssten (Plug-in Komponenten oder der Hauptanwendung)
- Bietet weitere **Schnittstellen** an, um den Zugang zu allen möglichen Funktionalitäten zu garantieren

Beteiligte der Plug-in-Architektur (3a)

Plug-in Komponenten:

- bestehen aus **Plug-in-interface**, **Plug-in-loader** und den **konkreten Plug-ins** wie sie Michael eben beschrieben hat
- **sind unabhängig von einander** (benutzen nur das Plug-in-interface und den Plug-in-loader von anderen plug-in Komponenten, aber rufen nie selber ihre konkreten Plug-ins auf)
- Erlauben das **Erweitern** des Systems **um einen Aspekt**

Beitragende der Plug-in-Architektur (3b)

- Stellen die wichtigsten Schnittstellen, die Plug-in-interfaces, bereit (neben den Schnittstellen der Bibliothek)
- Kein Teil des Systems ist also von einem konkreten Plug-in abhängig, sondern nur von dessen Interface
- Neue konkrete Plug-ins benötigen also keine Veränderungen in anderen Teilen des Systems (machen also auch vorherige Tests nicht ungültig)
- Weil somit alle konkreten Plug-ins einer Plug-in Komponente alle die gleiche Schnittstelle benutzen, muss nur einmal ein Dummy-Klassen- oder Mock-Klassen Test durchgeführt werden

Wie entwickle ich eine Plug-in-basierte Anwendung?

1. **Identifikation** der Teile des Systems die als Plug-in Komponenten in Frage kommen
Aufteilung in zwei verschiedene Arten von Plug-ins: jedes Plug-in hat eine **andere Funktionalität** (image filters) oder die Funktionalität ist dieselbe, aber das **Eingabe- oder Ausgabeformat** ist anders (import, export files)

Wie entwickle ich eine Plug-in-basierte Anwendung?

2. **Spezifikation** der Plug-in-interfaces der identifizierten Plug-in Komponenten: So **simpel wie möglich**
3. **Identifikation der Elemente** die **von mehr als einer Plug-in Komponente** oder der **Hauptanwendung verwendet werden**
4. **Spezifikation der Bibliotheks-Schnittstellen**

Wie entwickle ich eine Plug-in-basierte Anwendung?

5. Design der Hauptanwendung mit den Basis-Funktionalitäten
 6. Design der ganzen Bibliothek
 7. Design der konkreten Plug-ins
 8. Implementation der Hauptanwendung
 9. Implementation der Bibliothek
 10. Implementation der Plug-in Komponenten
- 8,9 und 10 kann parallel gemacht werden

Beispiel SLC

Die Portlets werden von der Hauptanwendung geladen

Die Hauptanwendung dirigiert den Datenfluss zu den einzelnen Portlets

Die Portlets sind Plug-ins, die während der Laufzeit geladen werden

The screenshot shows a Microsoft Internet Explorer window titled "SLC - Microsoft Internet Explorer". The address bar displays "https://slc.mathematik.uni-ulm.de". The browser interface includes a menu bar (Datei, Bearbeiten, Ansicht, Favoriten, Extras) and a toolbar with navigation and utility icons. Three callout boxes with light blue backgrounds and black outlines point to specific areas of the page:

- 1. zur Eingabe
bereites Portlet**: Points to a login form on the left side of the page. The form contains two input fields labeled "Login" and "Passwort", and a button labeled "Anmelden".
- 2. aktives
Portlet**: Points to a central content area titled "Administrative Tagesmeldungen". Below the title are several blue hyperlinks: "Zur Sauberkeit in den Pools in O27/211-213", "Chaos-Seminar", "Vorlesungsvorbesprechung Web Services", and "Stromabschaltungen 2003".
- 3. Zur Eingabe
bereites Portlet**: Points to a bottom-left section titled "Lehrangebot". Below the title is a button labeled "Vorlesungsverzeichnis".

At the bottom of the page, there is a footer with the text "slc@mathematik.uni-ulm.de ;-)" and a partially visible line of text "Bitte Name und Paßwort eingeben." above the "Administrative Tagesmeldungen" section.

Plug-in-basierte GUIs

- Spezielle Form der Plug-in-basierten Anwendungen
- Anwendung der **Plug-in-basierten Anwendungsentwicklung** auf GUIs
- Hier ist die **Hauptwendungen eine GUI**, die aus Plug-ins besteht
- Sie **implementiert** nur den **statischen Part** der GUI
- und bindet die GUI Plug-ins ein

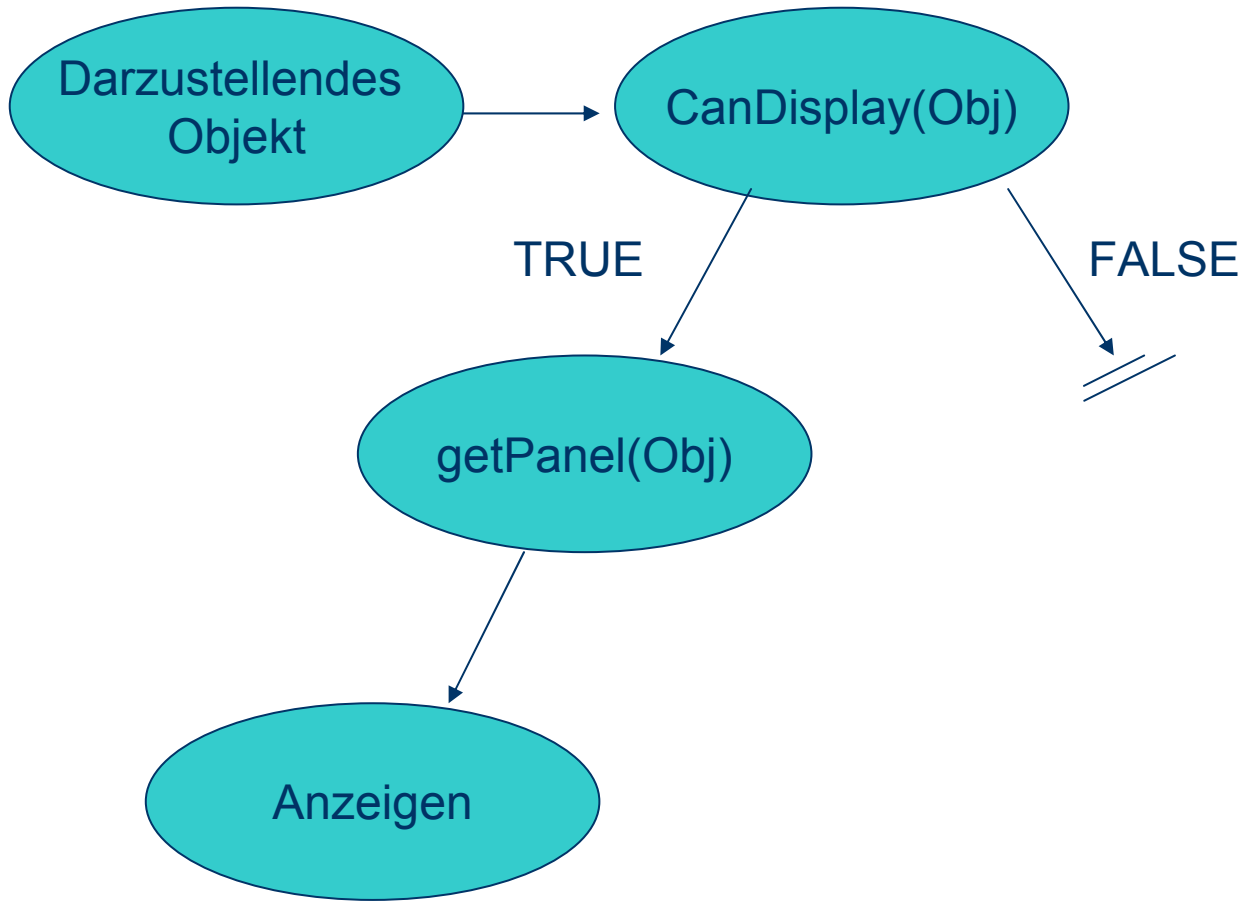
Mögliche GUI Plug-in Typen

- **View** plug-ins (display panels)
- **Preference** plug-ins (preference panels)
- **Command** plug-ins (menu, icon buttons)
- **Adjustment** plug-ins (adjustment panels)
- **Import/Export** plug-ins (import/export dialogs)
- **Info** plug-ins (info dialogs)
- **Help** plug-ins (help dialogs)

Beispiel View plug-ins (display panels)

- **Motivation:** Anwendungen sollten jede Art von Dokument Typen anzeigen können
- **ViewPlugIn:** bietet die „voting method“ `canDisplay()` an. Gibt ein Feld (Panel) zurück, in dem dann über ein `TRUE`-plug-in, das Objekt angezeigt wird
- **ViewPlugInPanel:** lädt mögliche View plug-ins mit `PlugInLoader`. Stellt Objekt über `getModel()` bereit. `setModel()` sucht ein Plug-in, welches das Objekt anzeigen kann
- **ConcreteViewPlugIn:** beliefert das `ViewPlugIn` interface, und kann einen ObjektTyp anzeigen

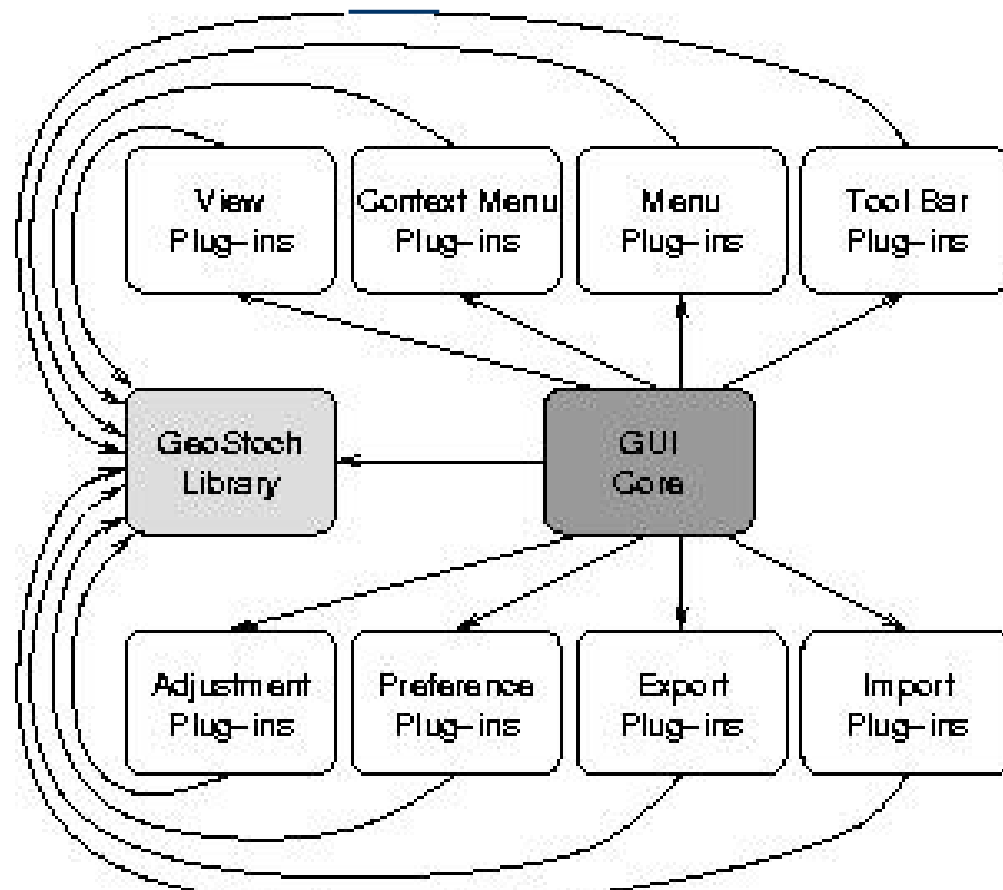
Beispiel View plug-ins



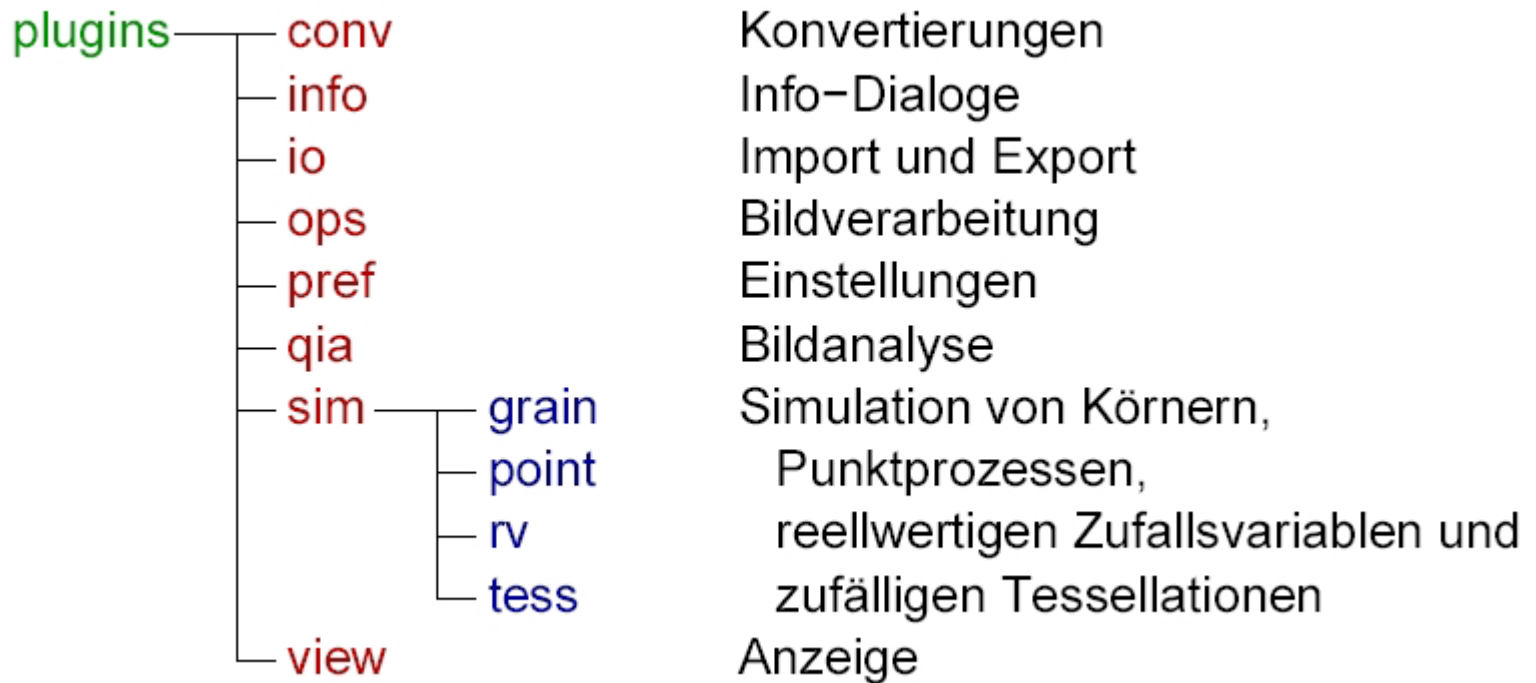
Die GeoStoch GUI

- Die **GeoStoch GUI** hat die **gleiche Architektur** wie eben beschrieben
- Selbst **kleinste Teile** sind **Plug-ins**
- Auf Basis des **Frameworks** entwickelt
- Hauptanwendung hat nur etwa **240 Code-Zeilen**
- Die **GUI** kann **durch** Hinzunahme oder Entfernen von **Plug-ins** **konfiguriert** werden

GeoStoch-Plug-in-Architektur:



Die Plug-in Package Struktur der GeoStoch GUI



Displaying an imported Image

The screenshot shows the GeoStoch software interface with several windows and a menu. The main window displays a red and white pattern of overlapping circles. A smaller window titled "s890t6.by" shows a grayscale image of a porous structure. A menu is open, showing options for "Processing", "Analysis", and "Close". A speech bubble points to the menu, and another points to the "s890t6.by" window.

GeoStoch

File Simulation Windows Info

Germ Grain Model

Binary Image of a Ran...

s890t6.by

Distance Trans...

Distance Transform (Euclidean)

Processing ▸
Analysis ▸
Close

Fourier Transform
Inversion
Distance Transform

View plug-in
(Binäres Bild)

View plug-in
(random-set Bild)

Command plug-in

Validity Check of the Input:

The screenshot displays the GeoStoch software interface. The main window has a menu bar with 'File', 'Simulation', and 'Windows'. Below the menu is a toolbar with icons for file operations and simulation. A 'Germ Grain Model' dialog is open, showing 'Germ Process: Point Proc...' and 'Primary Grain: Random Grain ...'. A callout bubble points to the toolbar icons, stating 'menu- und toolbar plug in (Zwitter)'. Below this, a 'Primary Grain' dialog is open, showing 'Disc' as the selected shape and 'Radius: RandomVariable ...'. A callout bubble points to the 'Radius' field, stating 'Adjustment plug-in'. Below the 'Primary Grain' dialog, a 'Radius' dialog is open, showing 'Uniform' as the selected distribution and empty input fields for 'Lower Bound' and 'Upper Bound'. At the bottom, an 'Error' dialog box displays a red stop sign icon and the message: 'The lower bound must be a real number! The upper bound must be a real number!'. An 'OK' button is visible at the bottom of the error dialog.

GeoStoch

File Simulation Windows Info

Germ Grain Model

Germ Process: Point Proc...

Primary Grain: Random Grain ...

Ok Cancel

Primary Grain

Disc

Radius: RandomVariable ...

Ok Cancel

Radius

Uniform

Lower Bound:

Upper Bound:

Ok Cancel

Error

The lower bound must be a real number!
The upper bound must be a real number!

OK

menu- und toolbar
plug in (Zwitter)

Adjustment plug-in

Zusammenfassung:

- + **weniger Komplex** (wenig Abhängigkeiten)
- + unterstützt **parallele Entwicklung** und **Entwicklung durch Dritte**
- + **keine Änderungen in externen Modulen** nötig
- + Entwickler werden durch **frameworks** unterstützt
- - Plug-ins müssen gegenseitig unabhängig sein
 - + aber dadurch **klare Grenzen**
- - Viele kleine Module
 - auch notwendig für ein gut strukturiertes Design bei Anwendungen ohne Plug-ins
 - + **Gut-organisiert durch Pakete**

**Danke für die
Aufmerksamkeit**

Fragen???
