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

- Simples 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

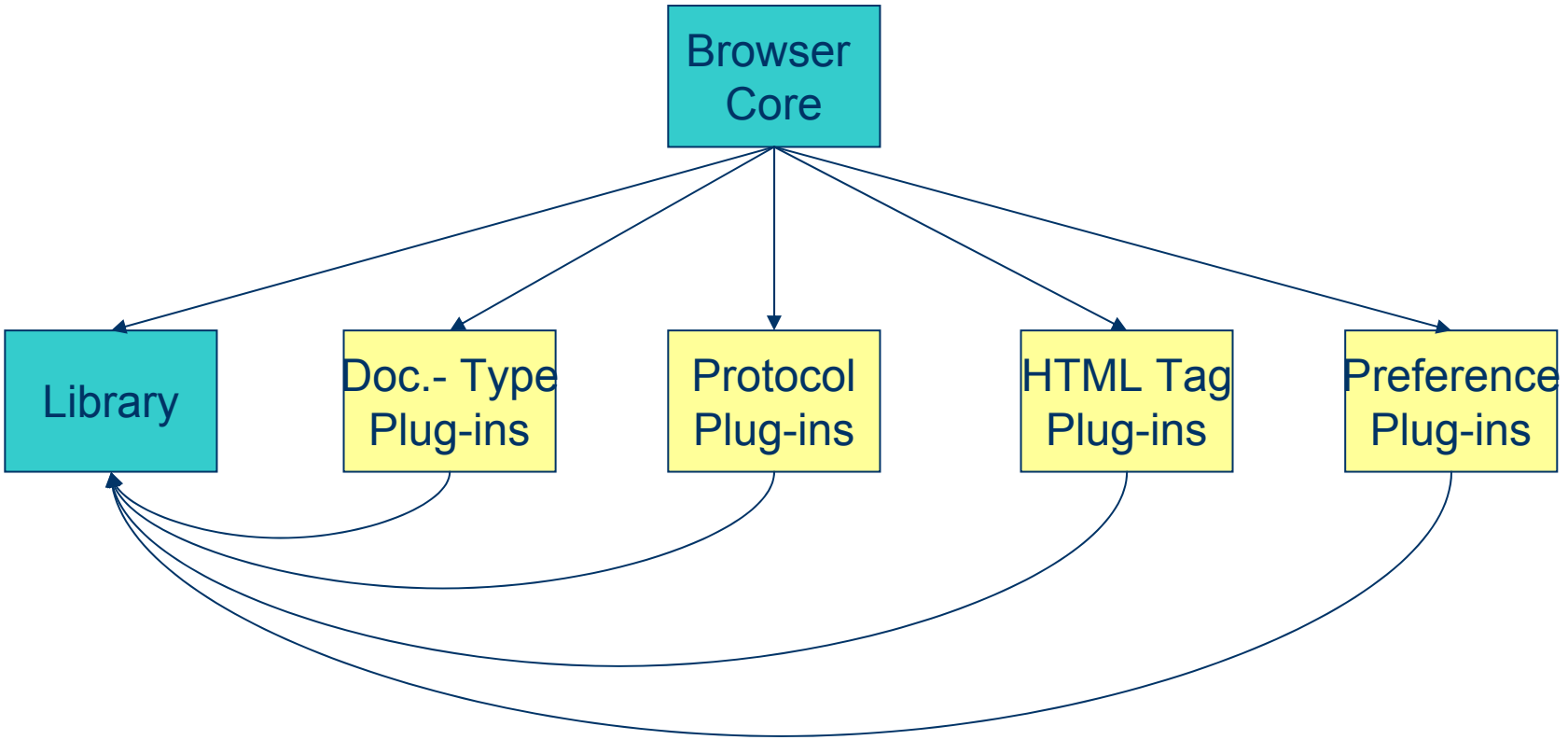
Was ist ein Plug-in?

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

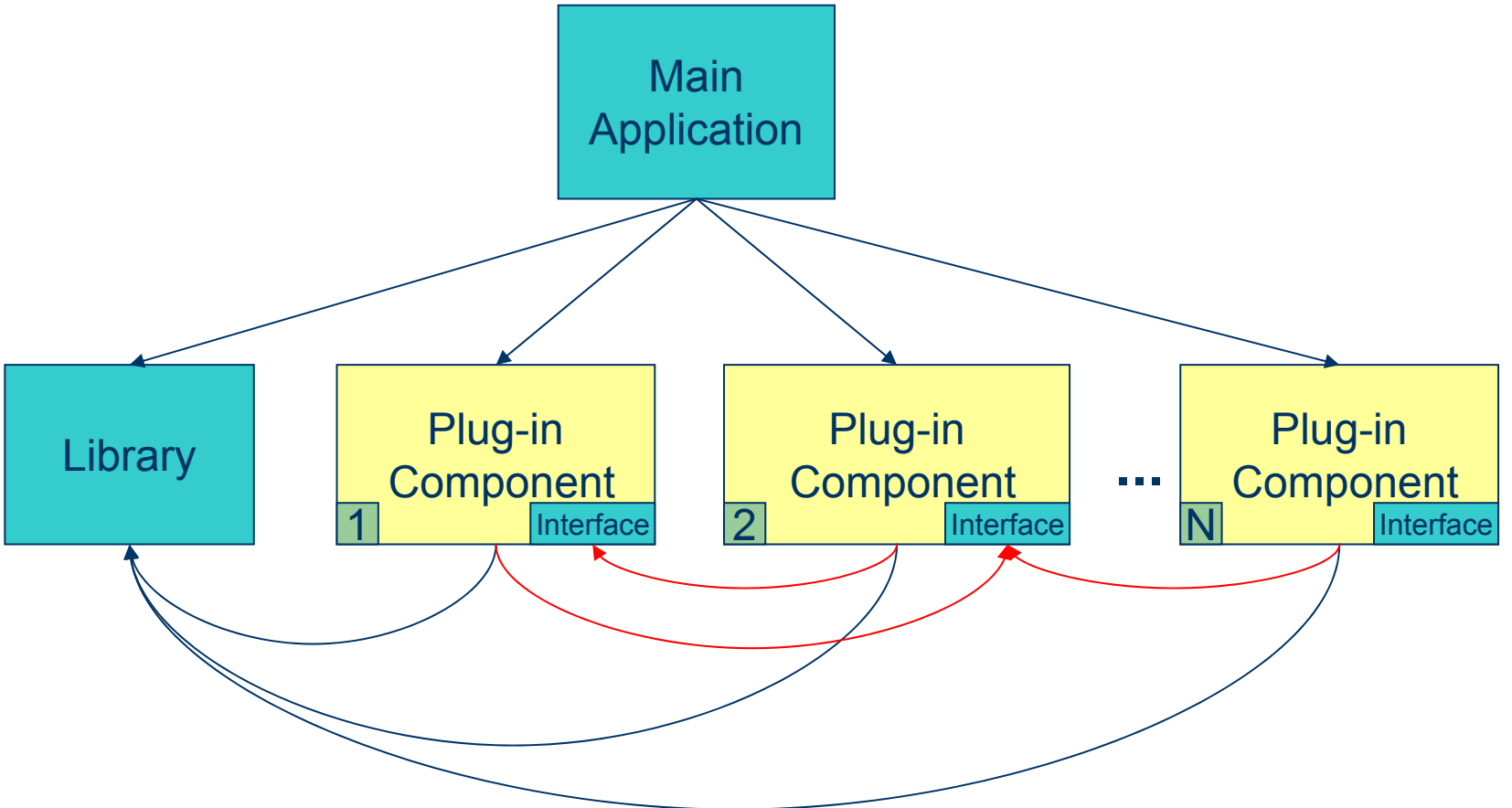
Plug-in-Architektur von einem Browser

- 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 seperat 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

Beiteiligte der Plug-in-Architektur (3b)

- Stellen die wichtigsten Schnittstellen, die Plug-in-interfaces, bereit (neben den Schnittstellen der Biblothek)
- 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

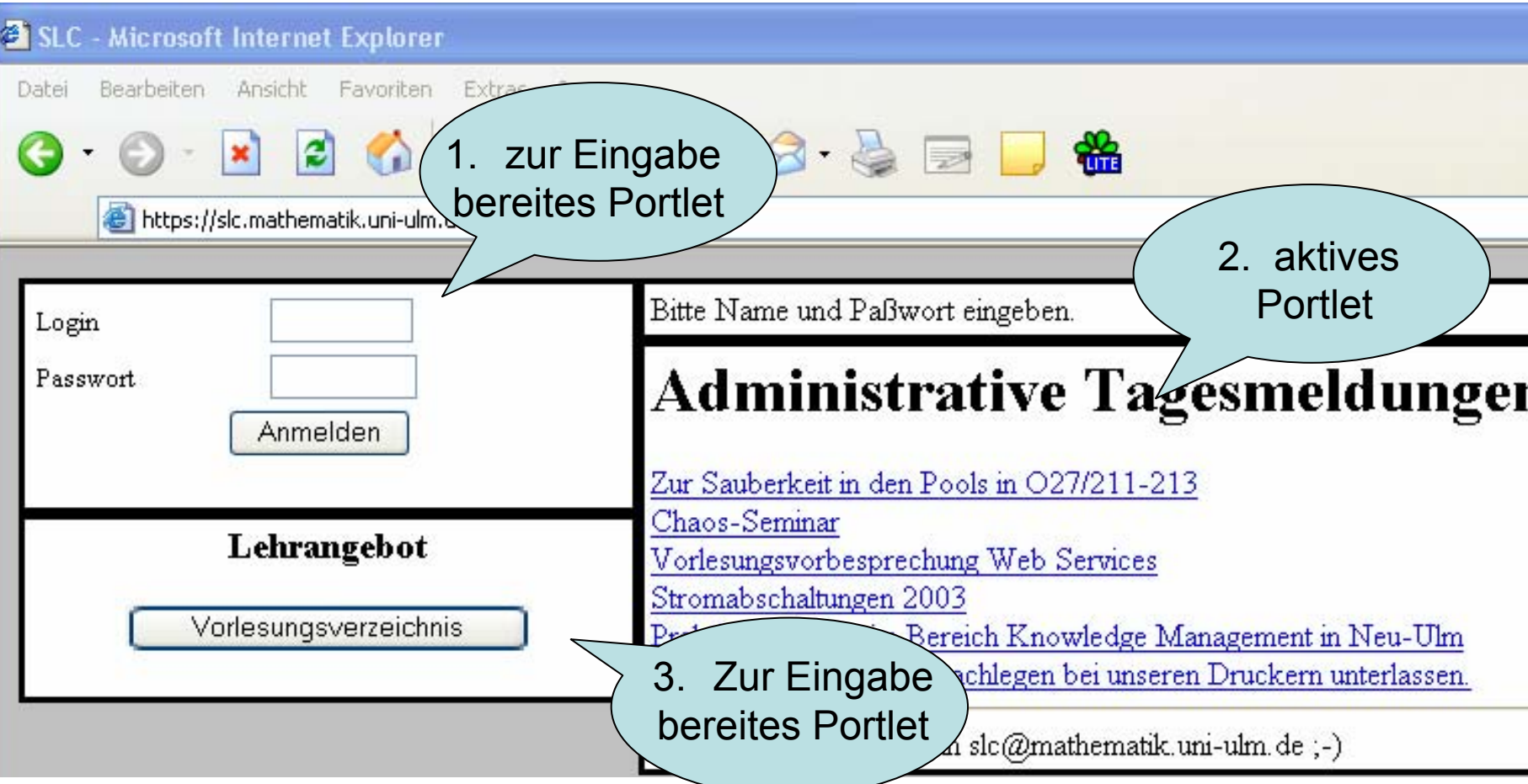
8,9 und 10 kann parallel gemacht werden

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



Plug-in-basierte GUIs

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

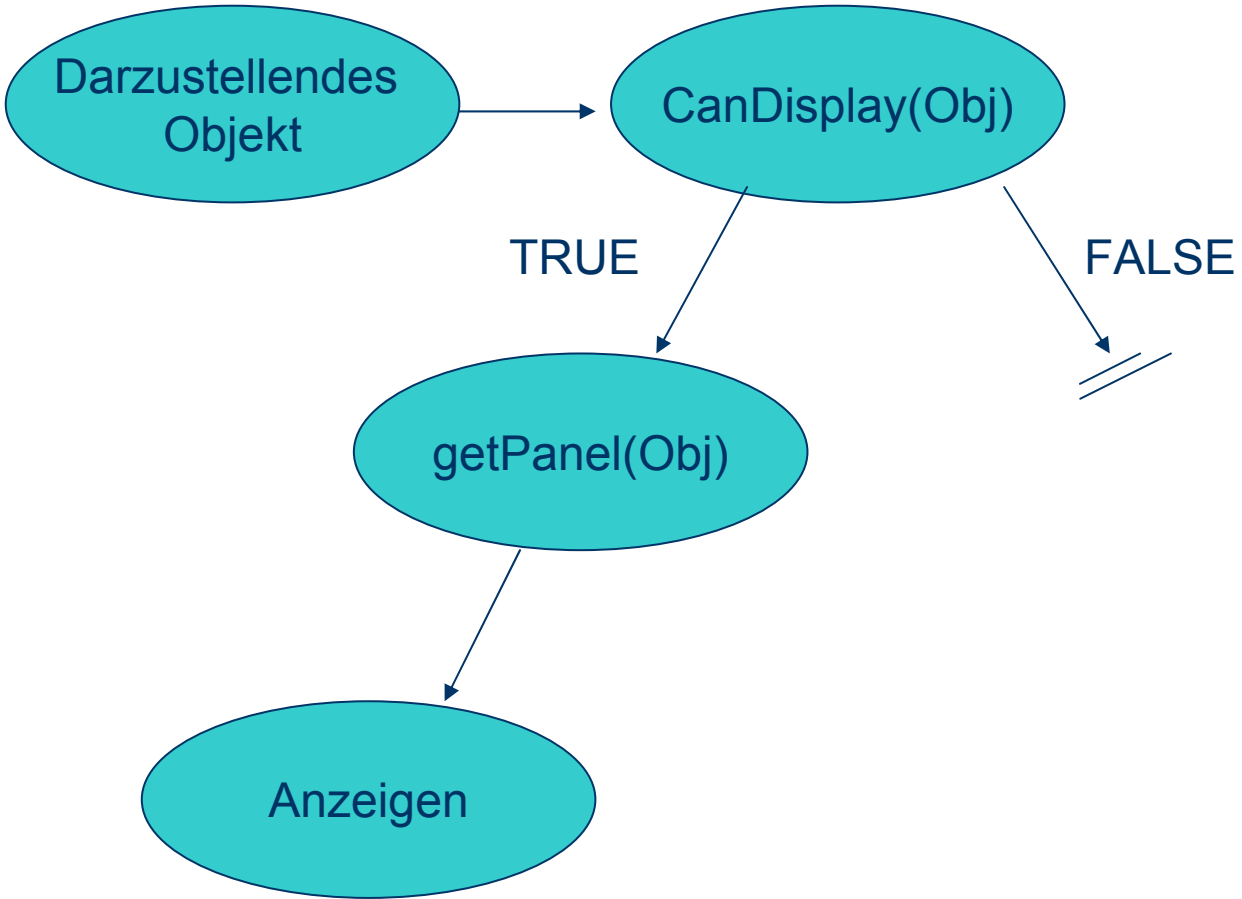
Mögliche GUI Plug-in Typen

- 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

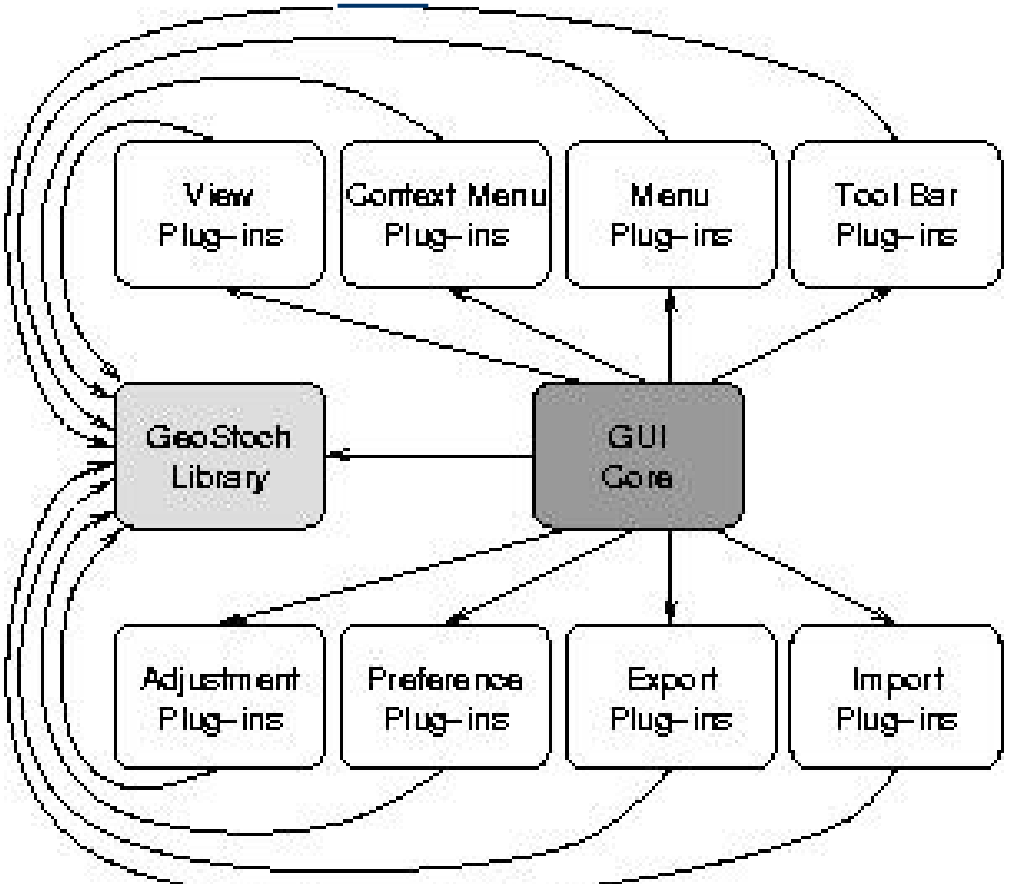
Beispiel View plug-ins



Die GeoStoch GUI

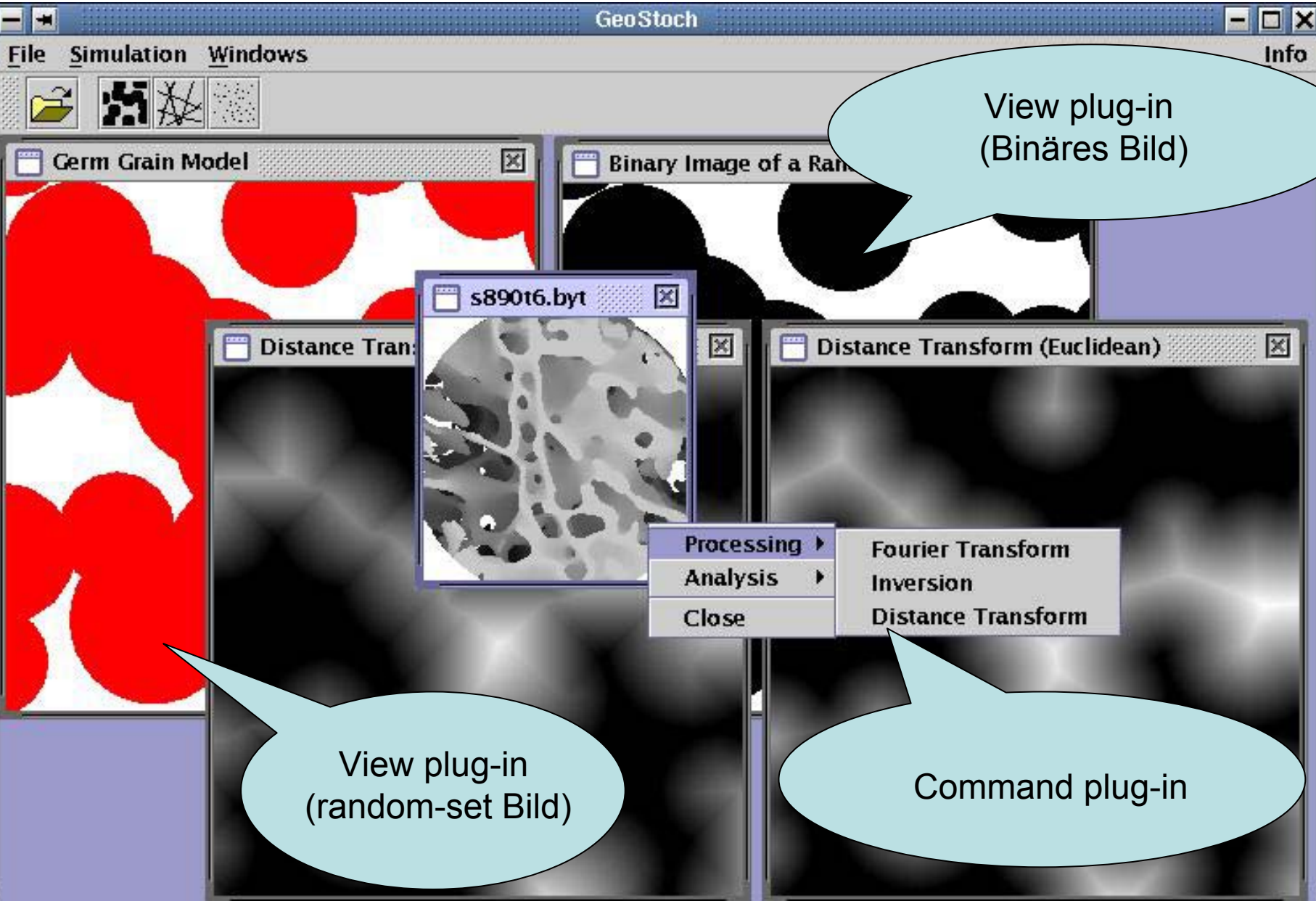
- Die GeoStoch GUI hat die gleiche Architektur wie eben beschrieben
- Selbst kleinste Teile sind Plug-ins
- Auf Basis des Frameworks entwickelt
- Hauptanwendung hat nur etwa 240 Code-Zeilen
- Die GUI kann durch Hinzunahme oder Entfernen von Plug-ins konfiguriert werden

GeoStoch-Plug-in-Architektur:

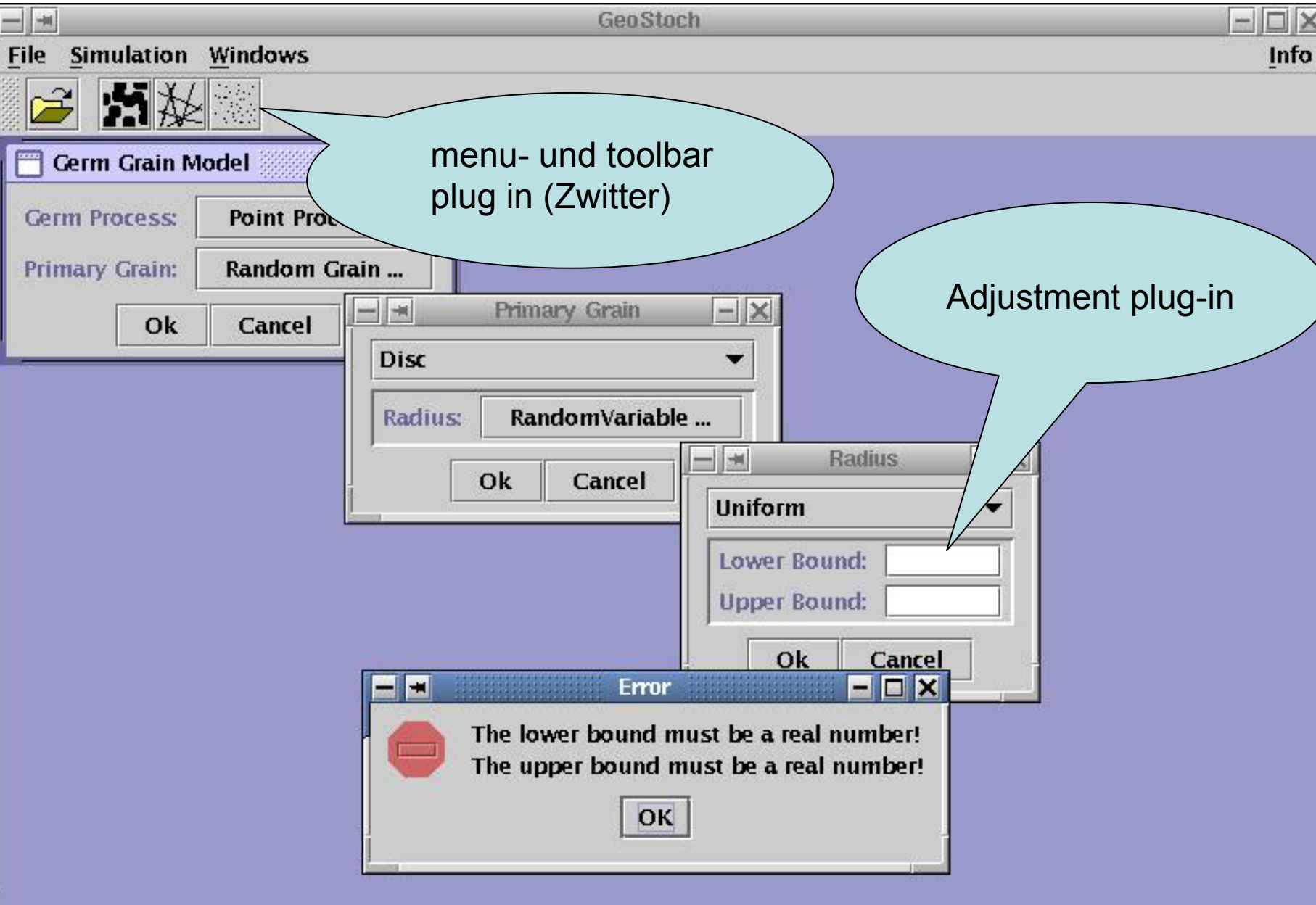


Die Plug-in Package Struktur der GeoStoch GUI

Displaying an imported Image



Validity Check of the Input:



Zusammenfassung:

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

Danke für die Aufmerksamkeit

Fragen???