Kapitel 5

Implementation

Nachdem die verschiedenen Konzepte der modellbasierten Bildverarbeitung erläutert wurden, seien in diesem Kapitel noch einige Aspekte der Implementation erwähnt, die den Rahmenbedingungen, unter denen die entwickelte Software angewandt werden soll, Rechnung tragen. Einige wichtige Aspekte wurden bereits in den Abschnitten 2.4 und 2.5 erwähnt. Hierzu gehört unter anderen die Tatsache, daß die zu erwartenden Benutzer im klinischen Umfeld in der Regel wenige Vorkenntnisse mit der Nutzung von Computern besitzen werden. Hinzu kommt, daß die Hardwareausstattung eher im unteren Preis- und Leistungsniveau einzuordnen ist. Bei dem Entwurf und der Umsetzung der Software sind solche Aspekte des zukünftigen Anwendungsbereichs zu berücksichtigen, damit eine sinnvolle Nutzung und damit der Einsatz der entwickelten Algorithmen möglich ist.

In den nachfolgenden Abschnitten 5.1 bis 5.3 wird beschrieben, wie durch den zugrunde gelegten Systementwurf versucht wurde, den Vorgaben des klinischen Umfelds Rechnung zu tragen. Erläutert werden die Auswahl der Programmiersprache, die Vorgabe der Softwarearchitektur und der Entwurf des Oberflächendesigns.

5.1 Programmiersprache: Java

Einer der wichtigsten Aspekte des Systementwurfs mit langfristigen Auswirkungen auf die Realisation der Software ist die Festlegung der Programmiersprachen, die zur Anwendung kommen sollen. Entscheidend für die Auswahl sind dabei sowohl die funktionalen Aspekte bezüglich der Performance und der Realisierungsmöglichkeiten der Algorithmen, aber auch die Rahmenbedingungen bezüglich der zu erwartenden Hardware- und Softwareumgebungen.

Bei der Auswahl der Programmiersprache wurde berücksichtigt, daß sowohl die Hardwarearchitektur als auch die verwendeten Betriebssysteme im klinischen Umfeld außerordentlich heterogen sind. Im Klinikum Benjamin Franklin kommen Unix-Workstations, PCs, Macintosh-Rechner und Linux-Rechner zum Einsatz. Als Betriebssysteme werden Windows 95, Windows 98, Windows NT, Windows 2000, Dec-Unix, Linux und 0S2 eingesetzt. Eine sinnvolle Beschränkung auf eine oder wenige Plattformen erscheint nicht ohne das Ausschließen von wichtigen, zukünftigen Nutzerkreisen möglich.

Im stationären Bereich und als Privatrechner der Ärzte sind Macintosh-Rechner und PCs am weitesten verbreitet, während im wissenschaftlichen Umfeld oft Unix-Workstations und Linux-Rechner genutzt werden. Da nicht erwartet werden kann, daß zusätzliche Hardware zur Nutzung einer neuen Software gekauft wird, muß bei der Auswahl der Programmiersprache eine angemessene Plattformunabhängigkeit gewahrt bleiben. Auf der anderen Seite sollten in der Programmiersprache Oberflächendesign und Event-Handling problemlos und stabil realisiert werden können. Die Verwendung einer objektorientierten Sprache ist daher auch in Hinsicht auf die Wartung und Weiterentwicklung naheliegend. Da bei der Auswertung der Bilddaten dreidimensionale Konstellationen von Oberflächen und Organen zu berücksichtigen

sind, wäre außerdem die Anbindung der Programmiersprache an Bibliotheken mit dreidimensionalen Rendering-Funktionen von großem Vorteil.

Für die Implementierung von Applikationen mit ähnlichen Anforderungen und Aufgabenstellungen werden derzeit vor allem die folgenden Programmiersprachen genutzt: C, C++, Delphi und Java. Die Sprache C kommt nicht in Betracht, da sie nicht objektorientiert ist und damit grundlegende Nachteile in der Wartbarkeit und Modularität gegenüber den objektorientierten Programmiersprachen aufweist. C++ ist zwar objektorientiert, muß jedoch für alle Betriebssysteme separat kompiliert werden. Dabei hat sich gezeigt, daß die Compiler der einzelnen Betriebssysteme nur bedingt kompatibel sind. Individuelle Anpassungen wären somit notwendig. Darüber hinaus ist das Oberflächendesign und das Event-Handling deutlich komplexer als es bei anderen objektorientierten Sprachen der Fall ist. Delphi ist objektorientiert, bietet eine Schnittstelle zu DirectX und erlaubt ein einfaches und sauberes Erstellen von Benutzeroberflächen. Eine Anwendung von Delphi-Programmen ist jedoch derzeit nur auf Windows-Rechnern möglich. Die notwendige Plattformunabhängigkeit ist damit nicht gegeben.

Einen guten Kompromiß zwischen den Anforderungen und den Möglichkeiten der Programmiersprache stellt Java dar. Java ist plattformunabhängig und kann auf allen erwähnten Betriebssystemen laufen. Oberflächendesign und Event-Handling können in Java durch umfangreiche Bibliotheken und eine saubere objektorientierte Struktur stabil, benutzerfreundlich und mit geringem Arbeitsaufwand realisiert werden. Durch das zielgerichtete Ausrichten der Sprache auf zukünftige Anwendungsbereiche durch Sun sind bereits im JDK-Standard interessante Funktionen wie Web-Browser und CORBA-Anbindungen realisiert, die die Entwicklung der Algorithmen für das klinische Umfeld durchaus erleichtern und beschleunigen. Darüber hinaus existiert zusätzlich eine 3D-API für Java, die eine direkte Schnittstelle zwischen Java und OpenGL herstellt. Dadurch können Segmentierungen und Daten bei einem geringen Implementierungsaufwand dreidimensional mit Hardwareunterstützung dargestellt werden. Nach dem Einbau einer 3D-Graphikkarte konnten selbst große Datensätze mit umfangreichen Segmentierungen ohne zusätzlichen Aufwand in Echtzeit visualisiert werden.

Einschränkend muß jedoch angemerkt werden, daß zumindest derzeit die Anwendbarkeit von Java-Applikationen von der verwendeten Version der Basisklassen JDK und dem Stand der kontinuierlich laufenden intensiven Entwicklungsbemühungen durch Sun bestimmt wird. Entscheidend ist, ob für das jeweilige Betriebssystem die aktuelle Runtime-Environment bereits entwickelt wurde. Wurde die Applikation mit dem Java-Standard 2.1 entwickelt, kann es durchaus vorkommen, daß für Linux oder ein anderes Betriebssystem lediglich die vorangegangene JDK-Version existiert. In solchen Fällen ist ein fehlerfreies Laufen der Software nicht gewährleistet. Es ist jedoch lediglich eine Frage der Zeit, wann die benötigten Basisklassen zur Verfügung stehen.

Neue Versionen der Java-Bibliothek JDK werden in der Regel zunächst für das Sun-Betriebssystem Solaris und für Windows entwickelt. Die Basisklassen für die übrigen Systeme werden zum Teil mit einer beträchtlichen Zeitverzögerung veröffentlicht. Da jedoch immer auch ein beträchtlicher Zeitraum zwischen der Entwicklungsphase und dem tatsächlichen Einsatz von Software im klinischen Umfeld liegt, ist dieser Nachteil als nicht gravierend einzuschätzen. Liegen die jeweiligen Runtime-Environments vor, ist die Kompatibilität sichergestellt, da das Übertragen des Source-Codes in den ausführbaren Code direkt von dem angepaßten Interpreter vorgenommen wird. Auf diese Weise kann die entwickelte Applikation auf allen Plattformen wie Unix, Linux, Macintosh und PC ausgeführt werden. Übliche Einsatzplattform wird in der Regel jedoch Windows sein.

Der Einwand, daß Java als Interpreter-Sprache zu langsam für Bildverarbeitungssoftware wäre, wird durch die aktuellen Weiterentwicklungen zur Beschleunigung von Java entkräftet. Durch angepaßte Compiler kann bereits ein Java-Code erzeugt werden, der nahezu die Performance von C erreicht. Bereits auf

einem INTEL Pentium 2, 400 Megahertz Taktfrequenz und 128 MByte RAM ist die Performance der von uns entwickelten Java-Applikation kaum noch zu unterscheiden von einer Delphi-Anwendung.

Die Realisation in Form eines Java-Applets, die noch flexibler wäre als eine Java-Applikation, da in diesem Fall lediglich ein Browser für die Nutzung installiert werden müßte, wurde nicht angewandt, da die Performance deutlich schlechter wäre als bei einer Applikation und da außerdem von einem Applet aus nicht auf die lokale Festplatte zugegriffen werden könnte.

5.2 Client-Server-Architektur versus Offline-Verarbeitung

Gerade in Zeiten immer schneller werdender PCs und entsprechenden Fortschritten bei der Kommunikation zwischen Anwendungen über das Internet ist die Entscheidung zwischen einer Client-Server- oder einer Stand-Alone-Applikation mit Hinsicht auf die Performance und die Benutzerfreundlichkeit von großer Bedeutung.

Arbeitsabläufe im klinischen Umfeld zeichnen sich oft durch einen großen Zeitdruck aus. Dieser Zeitdruck resultiert weniger aus der Notwendigkeit zur schnellen Durchführung von aufeinander folgenden Arbeitsschritten, sondern wird eher durch die Quantität der durchzuführenden Aufgaben hervorgerufen. Als Folge wird neuen und zusätzlichen Tätigkeiten wie der computergestützten Auswertung von Bilddaten oft nur ein geringer zusätzlicher Arbeitsaufwand zugestanden. Wird ein bestimmter Zeitaufwand überschritten und besteht kein erhöhtes wissenschaftliches Interesse der verantwortlichen Autoritäten, ist damit zu rechnen, daß eine entwickelte Software nicht zur Anwendung kommt. Während der Arbeitsaufwand ein wichtiger Faktor für die Akzeptanz von Software darstellt, ist der Zeitpunkt, bis zu dem die Ergebnisse vorliegen müssen, durchaus flexibel und kann einige Tage nach der eigentlichen Untersuchung liegen. Ein Zeitraum von mehreren Tagen bis Wochen zwischen den Aufnahmen und der Therapie ist selbst bei den heutigen Verfahren durchaus üblich und akzeptabel.

Auf der anderen Seite sind Bildverarbeitungsalgorithmen oft mit einem hohen Rechenaufwand verbunden. Gerade die im Rahmen des Forschungsvorhabens anzuwendende semiautomatische Segmentierung und Registrierung der Bilddaten ist mit einer Vielzahl von Berechnungen für jeden einzelnen Bildpunkt verbunden. Bei Bilddatensätzen mit mehr als zehn Datenschichten und einer minimalen Auflösung von 256 mal 256 Pixeln benötigen auch die immer schneller werdenden PCs und Workstations Rechenzeiten von mehreren Minuten, gegebenenfalls sogar Stunden.

Unter Berücksichtigung dieser Aspekte bietet sich ein Realisierungskonzept an, bei dem der Benutzer lediglich die notwendigen Eingaben interaktiv vornimmt, während rechenaufwendige Verarbeitungsschritte offline ohne Beaufsichtigung ausgeführt werden. Daraus resultierten die Forderungen, Routineschritte weitgehend zu automatisieren und die zur Verfügung stehende Arbeitszeit auf absolut notwendige Eingaben zu beschränken. Um diese beiden konträren Forderungen in Einklang zu bringen, können zwei unterschiedliche Strategien verfolgt werden:

1. Rechenintensive Arbeiten werden in voneinander unabhängige Teilaufgaben unterteilt und anschließend parallel auf einem Server-Cluster abgearbeitet. Durch die Parallelisierung der Berechnungen soll eine Beschleunigung erreicht werden, die die Wartezeiten für den Benutzer auf ein akzeptables Maß reduziert.

2. Zunächst werden die unverzichtbaren interaktiven Arbeiten durch den Benutzer vorgenommen. Anschließend werden die rechenintensiven Berechnungen offline ohne Aufsicht des Benutzers automatisch ausgeführt. Nach Abschluß der Arbeiten kann der Benutzer sich die Ergebnisse anschauen und gegebenenfalls modifizieren.

Ein wichtiger Aspekt, der bei beiden Ansätzen beachtet werden sollte, ist die Minimierung der tatsächlich durch den Benutzer investierten Zeit. Eine Beschleunigung der Arbeiten durch eine Parallelisierung der Rechenschritte macht nur dann Sinn, wenn der Benutzer dadurch nicht mehr lange untätig am Rechner warten muß. Demgegenüber stellt eine Offline-Berechnung nur dann kein Problem dar, wenn nach Abschluß der automatischen Berechnungen keine langwierige Nachbearbeitung der Ergebnisse notwendig wird.

Die parallelisierte Berechnung durch einen Server-Cluster könnte folgendermaßen realisiert werden: Der Benutzer lädt in einem Java-Client die Bilddaten ein und führt die einzelnen Schritte der Bildverarbeitungspipeline aus. Bei Programmteilen, die sehr rechenintensiv sind, werden die Daten und Berechnungsschritte in voneinander unabhängige Teilaufgaben unterteilt. Daten und Berechnungsvorschrift werden vom Client aus über das Netzwerk auf mehrere Rechner (Server) verteilt, die zuvor als Berechnungsserver definiert wurden. Auf diesen Servern sind die notwendigen Funktionsbibliotheken in C, C++ oder Java installiert. Nach Abschluß der Berechnungen auf den Servern werden die Ergebnisse zum Client zurück geliefert und zu einem Gesamtergebnis zusammengefaßt. Durch eine sinnvolle Parallelisierung der Arbeitsschritte und das Verteilen auf unter Umständen deutlich schnellere Rechner kann die Bearbeitungszeit für Berechnungsschritte deutlich reduziert werden. Je mehr Server zur Verfügung stehen, um die Berechnungen durchzuführen und je mehr Schritte parallelisiert werden können, desto größer ist der zu erwartende Zeitgewinn. Voraussetzung für einen solchen Zeitgewinn ist jedoch, daß die bei den Berechnungen gesparte Zeit nicht durch den zusätzlichen Datentransfer, die Koordination der Verteilung und der anschließenden Integration der Ergebnisse wieder verloren wird.

Um diesen logistischen Zeitverlust abschätzen zu können, muß zunächst spezifiziert werden, wie die Parallelisierung vorgenommen und wie die Kommunikation mit den Servern hergestellt wird. Die Parallelisierung der Teilaufgaben an sich kann in Java problemlos über das Starten einzelner Threads realisiert werden. Diese Aufteilung der Aufgaben muß dementsprechend während der Programmierarbeiten konzeptioniert und programmiert werden. Für die Realisierung der Kommunikation zwischen den Client-Threads und den Servern stehen mehrere Möglichkeiten zur Verfügung:

- **Socket-Kommunikation**: Zwischen Client und Server wird eine Socket-Verbindung hergestellt, durch die Daten und Befehle übertragen werden. Vorteil dieses Ansatzes ist eine schnelle Datenübertragung, da wenig Protokoll-Overhead den Datentransfer abbremst. Nachteil dieser Kommunikationsform ist die Tatsache, daß auf diese Weise nur eine Low-level-Kommunikation möglich ist. Alle erforderlichen Koordinationsschritte wie das Abrufen der Server, untersuchen ob jeder einzelne Unterprozeß bereit ist, wie schnell der Server ist, wie hoch die jeweilige Auslastung derzeit ist und vieles mehr muß zusätzlich implementiert werden. Bei komplexen Daten- oder Architekturstrukturen müßte ein hoher Implementierungsaufwand betrieben werden, bevor eine verteilte Berechnung möglich ist. Erschwerend kommt bei unterschiedlichen Betriebssystemen hinzu, daß die Kommunikation für jedes System separat entwickelt werden muß.

- **Remote Procedure Calls (RPC)**: Bei einer verteilten Berechnung mittels RPCs würden zunächst die Daten auf die jeweiligen Server per ftp oder andere Protokolle kopiert werden. Anschließend würden die RPCs die entsprechenden Bearbeitungsalgorithmen aufrufen. Nach Abschluß der einzelnen Prozesse müssen die Ergebnisdaten zurück kopiert und kombiniert werden. Die Realisierung der Kommunikation über Remote Procedure Calls weist weitgehend ähnliche Vor- und Nachteile auf wie die Socket-Kommunikation.

- **CORBA:** CORBA ist eine sogenannte Middleware, die die Kommunikation zwischen Client-Anwendungen und Serverdiensten realisiert. Sie beinhaltet eine umfassende Infrastruktur zur Konvertierung und Abstraktion von Datenströmen, um die Koordination und Portabilität der

Verarbeitungsschritte für verschiedene Plattformen zu gewährleisten. CORBA realisiert das Interface und die Kommunikation bei einer Client-Server-Architektur für eine Vielzahl von Hardware, Betriebssystemen, Sprachen und Komponenten. CORBA-Schnittstellen sind bereits für alle gängigen Architekturen und Programmiersprachen realisiert worden und ist daher als plattformunabhängig anzusehen. Während der Client in Java programmiert ist, könnten die Server-Anwendungen in C oder C++ realisiert werden. Die Schnittstellen zwischen dem Client oder Server und dem zentralen Verteilserver, dem sogenannten Objekt Request Broker (ORB) wird mittels IDL (interface definition language) realisiert (siehe Abbildung 5.1). Das Verteilen der Clienttasks, Abwarten der Antworten, Exceptionhandling und Weiterleiten der Ergebnisse wird automatisch von dem Objekt Request Broker übernommen. Abbildung 5.1 zeigt beispielhaft die Struktur von Client, den ORBs und den Servern, wie sie für die gegebenen Aufgabenstellung realisiert werden könnte. Die Kommunikation zwischen den einzelnen ORBs wird über das Internet Inter-ORB Protokoll (IIOP) realisiert, das etwas langsamer als TCP/IP ist. CORBA ist eine objektorientierte Sprache und wird bereits in einigen Standardprogrammen wie beispielsweise dem Netscape Communicator eingesetzt und erfährt dadurch eine immer breitere Akzeptanz. Darüber hinaus bietet Netscape verschiedene Tools an, mit denen Java-Programmcode direkt in CORBA übersetzt werden kann. Der Vorteil dieses Ansatzes ist der geringe zusätzliche Programmieraufwand und das Zurückgreifen auf eine bereits getestete und sehr vollständige Schnittstelle zwischen Server- und Client-Anwendungen.

- **DCOM:** DCOM entspricht in seiner Funktionalität dem CORBA-Ansatz, ist als eine Microsoft Entwicklung jedoch auf Betriebssysteme von Microsoft beschränkt. Eine Plattformunabhängigkeit ist daher nicht gegeben.
- **RMI** (Java Remote Method Invocation): Java bietet ein eigenes Protokoll an, mit den Java-Applikationen andere Java-Applikationen aufrufen können. Dieses Verfahren bleibt jedoch auf Java-Anwendungen beschränkt.

Unter Berücksichtigung der Rahmenbedingungen im medizinischen Umfeld und den beschränkten Ressourcen im wissenschaftlichen Sektor stellt die Realisierung der Client-Server-Architektur unter Verwendung der CORBA Middleware die mit Abstand beste Lösung dar. Abbildung 5.1 veranschaulicht die grundsätzliche Architektur für einen auf CORBA basierenden Systementwurf.

Erste Tests mit CORBA haben jedoch gezeigt, daß die verteilte Berechnung nur bedingt eine Zeitersparnis durch die Parallelisierung bietet. In einzelnen Fällen ist durch den Datentransfer sogar mit einem erhöhten Rechenaufwand zu rechnen. Der Zeitaufwand für den Transfer von Daten über die CORBA-Schnittstellen vom Client zum Server und zurück resultiert weitgehend aus den notwendigen koordinativen Funktionsaufrufen. Minimale Transferzeiten selbst ohne größere Datenpakete lagen bei Tests innerhalb des vergleichsweise schnellen hausinternen Netzes im Sekundenbereich. Ein effizienter Einsatz der verteilten Berechnung wäre gerade bei den immer schneller werdenden PCs nur dann sinnvoll, wenn ein großer Cluster von sehr schnellen Rechnern zur Verfügung steht, die Netzwerkverbindungen sehr schnell sind und das Verhältnis von erforderlichem Rechenaufwand zu übertragenen Daten möglichst groß ist. Da diese Anforderungen bei dem gegebenen Projekt nicht in jedem Fall gegeben sind, wurde bei der Entwicklung der Software zwar eine Schnittstelle für einen CORBA-Middlelayer offen gehalten, eine tatsächliche Anbindung an einen Server-Cluster jedoch nicht realisiert.

Statt dessen bot sich der zweite Ansatz als eine gute Alternative an, den interaktiven Arbeitsaufwand auf ein Minimum zu reduzieren und die aufwendigen Berechnungen ohne Beaufsichtigung durch den Benutzer offline durchzuführen. Dadurch wurde sowohl der zusätzliche Implementierungsaufwand für die Schnittstelle zwischen Server und Client als auch für die gegebenenfalls komplexe Koordination der unterschiedlichen Threads vermieden. Andererseits hat sich mit fortschreitenden Projektarbeiten gezeigt,

daß die Geschwindigkeitsgewinne durch eine verbesserte Hardware so beträchtlich sind, daß der zu erwartende zusätzliche Gewinn durch die Realisierung eines Server-Clusters in jedem Fall gering ausfallen würde. Es wurde daher die einfache, aber angemessene Softwarearchitektur einer Standalone-Applikation zugrunde gelegt.

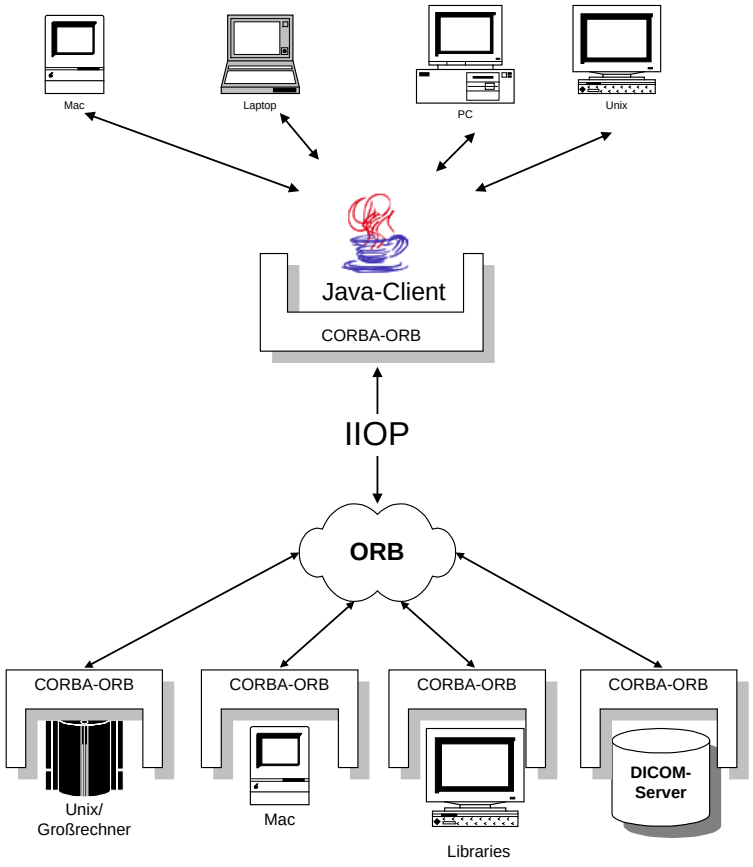

Abbildung 5.1: Mögliche Client-Server-Architektur für eine Bildverarbeitungssoftware mit CORBA

Während der Client in Java realisiert ist und dadurch auf unterschiedlichen Plattformen laufen kann, werden die Server-Anwendungen durch einen CORBA Object Request Broker gekapselt. Dadurch können die Server in unterschiedlichen Sprachen und auf verschiedenen Betriebssystemen realisiert werden. Als Kommunikationsprotokoll wird IIOP benutzt.

5.3 Oberflächendesign

Unabhängig von der tatsächlichen Realisierung der funktionalen Programmteile ist das Oberflächendesign an die Bedürfnisse und Fähigkeiten der zukünftigen Benutzer anzupassen. Oft ist es die Benutzerfreundlichkeit, die entscheidet, ob sich eine entwickelte Software durchsetzt oder nicht.

Solange die Software sich im Entwicklungsstatus befindet, werden sich die Benutzer hauptsächlich aus wissenschaftlichen Mitarbeitern mit Ingenieursausbildung zusammensetzen. Sobald jedoch eine routinemäßige Anwendung der Software für die Therapie von Augentumoren stattfindet, wird in erster Linie medizinisch ausgebildetes Personal im klinischen Umfeld die Auswertungen der Bilddaten vornehmen. Diese Vorgabe bringt spezifische Anforderungen an die Benutzerfreundlichkeit der entwickelten Bildverarbeitungssoftware mit sich. Im klinischen Umfeld muß zumindest derzeit noch mit Benutzern gerechnet werden, die einen vergleichsweise geringen Erfahrungshorizont bezüglich der Nutzung von Computern

mitbringen. Oberflächendesign und Benutzerführung müssen dementsprechend möglichst einfach gestaltet werden, so daß auch bei geringen Vorkenntnissen die sinnvolle Nutzung der Software möglich ist.

Bei dem grundsätzlichen Design der Software wurde darauf geachtet, daß der Benutzer so weit wie möglich Kenntnisse über die Bedienung von anderen Softwarepaketen auf die Bildverarbeitungssoftware übertragen kann. Aus diesem Grund wurde das grundsätzliche Layout der entwickelten Software „JDisplay“ an Standardprogrammen wie Photoshop und Extreme 3D ausgerichtet. Es gibt ein „Hauptfenster“, in dem die Bilddaten von einem oder mehreren Datensätzen dargestellt werden (siehe Abbildung 5.2). In dem sogenannten „Tool-Fenster“ kann der Benutzer die Bildverarbeitungsalgorithmen auswählen, konfigurieren und nutzen. Anhand des Navigations-Fensters kann der Benutzer durch den Datensatz navigieren und einstellen, ob die Daten in zweidimensionalen Schichten, in dreidimensional ausrichtbaren Schichten oder durch ein dreidimensionales Rendering dargestellt werden sollen.

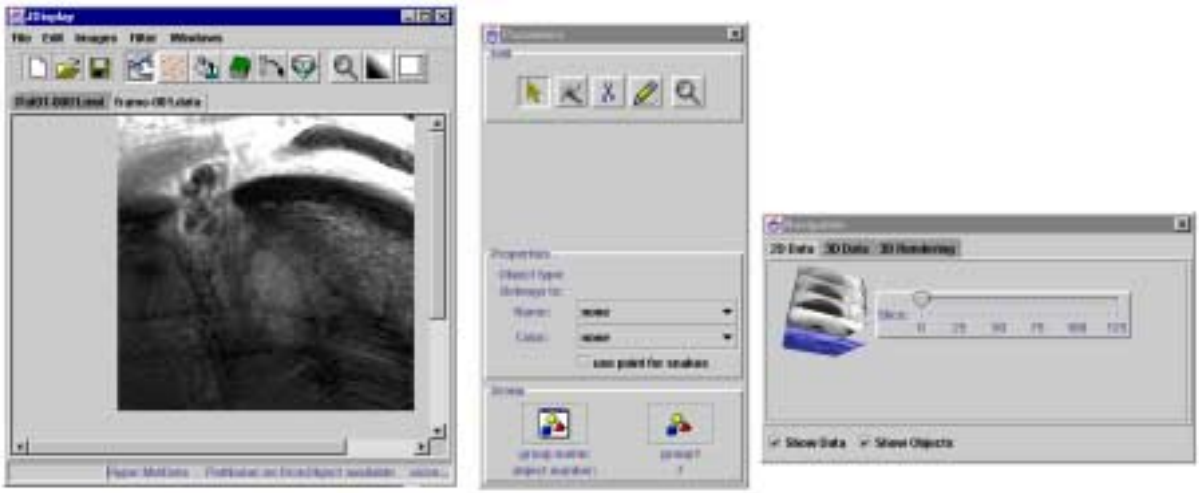

Abbildung 5.2: Die drei Hauptfenster der Bildverarbeitungssoftware JDisplay

In dem Hauptfenster ganz links werden die Bilddatensätze dargestellt. Durch Registerblätter kann zwischen den einzelnen Datensätzen gewechselt werden. In der Mitte ist das Tool-Fenster zu sehen, mit dem die Bildverarbeitungsalgorithmen konfiguriert und genutzt werden können. Mit dem Navigationsfenster ganz rechts kann der Benutzer durch den Datensatz navigieren. Außerdem kann er zwischen verschiedenen Visualisierungsmodi wählen. Er kann den Datensatz in seinen ursprünglichen Schichten oder als dreidimensionales Datenvolumen betrachten. Im dritten Registerblatt „3D Rendering“ können die Daten und die segmentierten Objekte im Rendering-Modus dreidimensional betrachtet werden.