



Universität Paderborn

sTeam - kooperative Wissensorganisation
Shared Whiteboard

Technische Dokumentation

Daniel Büse

Fakultät für Elektrotechnik, Informatik und Mathematik

Paderborn, Januar 2003

Inhaltsverzeichnis

1	Voraussetzungen für den Einsatz des Shared Whiteboard	2
1.1	Java	2
1.2	Betriebssystem	2
2	Implementierung	4
2.1	Die sTeam API	4
2.2	Das Package applicationlauncher	5
2.3	Aufbau der Packagehierarchie	5
2.4	Das Package steam.modules.whiteboard	7
2.5	Das Package steam.modules.whiteboard.control	8
2.5.1	Die Steuerungselemente	8
2.5.2	Kapselung globaler Variablen und Methoden	10
2.6	Das Package steam.modules.whiteboard.event	13
2.7	Das Package steam.modules.whiteboard.component	14
2.7.1	Die Zeichenflächen	19
2.7.2	Repräsentationen für die Standard-Objekte in sTeam	23
2.7.3	Die grafischen Objekte	25
2.7.4	Nicht bekannte Objekte	33

Kapitel 1

Voraussetzungen für den Einsatz des Shared Whiteboard

Im folgenden werden die technischen Voraussetzungen erläutert, die erfüllt sein müssen, um mit dem Shared Whiteboard Client arbeiten zu können. Zum einen ist es erforderlich die richtige Version des Java Runtime Environment installiert zu haben, andererseits wurde das Whiteboard nur auf einigen der vielen Betriebssystemen umfangreich auf die korrekte Funktion hin überprüft.

1.1 Java

Auf dem Zielrechner, auf dem das Whiteboard eingesetzt werden soll muss Java2 in der Version 1.3.1 installiert sein. Der Client wurde unter dieser Version entwickelt, eine fehlerfreie Ausführung kann nur unter der Version 1.3.1 erfolgen, da die Version 1.2 einige Methoden auf die innerhalb des Clients zugegriffen wird noch nicht implementiert hat. Zu Schwierigkeiten mit der neuesten Java Version kommt es bei der Handhabung der Textfelder, zudem können Inkompatibilitäten hinsichtlich des Drag and Drop entstehen, das sich die Java Runtime in diesem Bereich selbst noch einmal einen Schritt weiter entwickelt hat.

1.2 Betriebssystem

Durch die Verwendung von Java als Programmiersprache ist der Client nahezu Betriebssystemunabhängig. Der Client wurde auf den Systemen Windows XP, Sun OS, Mac Os X und Linux erfolgreich getestet. Für diese Systeme wurden auch einige native Teile programmiert, dazu gehören der Start eines Browsers, sowie der Start von lokal installierter Software zum Anschauen/Bearbeiten von Dokumenten. Die Systeme Sun Os und Linux erfordern eine spezielle Konfiguration des White-

boards durch das Anpassen der whiteboard.cfg um die entsprechenden lokalen Anwendungen zum Anschauen bzw Editieren der Dokumente starten zu können. Wie dies genau zu erfolgen hat lesen sie bitte in der aktuellen FAQ auf der Webseite <http://www.open-steam.org/clickclique> nach.

Kapitel 2

Implementierung

Im folgenden wird der technische Aufbau des Shared Whiteboard Clients beschrieben. Der Aufbau der API, auf die zur Kommunikation mit dem Server zurückgegriffen wird, wird näher erläutert. Anschließend wird auf den Aufbau des Whiteboard-Clients und die Strukturierung der einzelnen Klassen in Packages eingegangen. Die technische Umsetzung der wichtigsten Funktionen, die von den einzelnen Klassen zur Verfügung gestellt werden, wird anschließend beschrieben.

2.1 Die sTeam API

Die sTeam-API stellt bereits die grundlegenden Funktionen zur Kommunikation mit dem Server bereit. Die Klasse `SteamConnector` ermöglicht den Aufbau einer Verbindung zum Server. Für den Verbindungsaufbau müssen der Benutzername und das Passwort zur Authentifizierung des Benutzers an die entsprechende Methode übergeben werden. Der `SteamConnector` stellt die globalen Funktionen zur Verfügung. So beinhaltet er Methoden, mit denen neue Objekte auf dem Server angelegt werden können. Abhängig vom Typ des zu erstellenden Objektes müssen unterschiedliche Parameter übergeben werden, so z.B. bei dem Erzeugen eines neuen Room-Objektes der Name des anzulegenden Raumes. Zum Anlegen neuer Objekte greift der `SteamConnector` auf eine Implementierung von sogenannten "Factories" zurück, um so die Flexibilität und Erweiterbarkeit des Servers auch den Clients zu Verfügung zu stellen.

Für jeden auf dem Server existierenden Objekttyp hält die API eine Klasse bereit, die diesen Typ auf der Client-Seite implementiert und Zugriff auf die klassenspezifischen Funktionen bietet. Die Klassen `SteamObject`, `SteamContainer`, `SteamRoom`, `SteamDocument`, `SteamGroup`, `SteamUser` und `SteamDrawing` stellen dementsprechend die Implementationen auf Client-Seite für die Objekttypen `Object`, `Container`, `Room`, `Document`, `Group`, `User` und `Drawing` dar. Mit Hilfe dieser Klassen lassen sich z.B. die Attribute der Objekte auf dem Server setzen und

auslesen, Kopien der Objekte anlegen, das Löschen des Objektes initiieren und Objekte in andere Räume oder Container bewegen.

Die API setzt das Event-System des Servers auf der Client-Seite um. Durch spezielle Methoden in `SteamObject` ist es möglich, die verschiedenen Events zu abonnieren. Dazu muss die Klasse, die an dem speziellen Event interessiert ist, einen entsprechenden "Listener" bei dem API-Objekt registrieren. Die API übernimmt die Registrierung des Listeners bei dem entsprechenden Objekt auf dem Server. Wird der Event ausgelöst, bekommt die API eine entsprechende Nachricht vom Server und reicht diese als Instanz eines `SteamEvent` an die bei der API registrierte Listener-Instanz weiter. Diese `SteamEvent`-Instanz enthält neben dem Typ des aufgetretenen Events¹ eventspezifische Daten².

2.2 Das Package applicationlauncher

Dieses Package erlaubt es durch die Verwendung der Klasse `AppLaunch` für auf dem Server liegende Dokumente eine lokale Anwendung zu öffnen, die es erlaubt das Dokument zu verändern. Werden diese Änderungen abgespeichert, so sorgt die `AppLaunch`-Instanz dafür, dass die geänderten Daten automatisch auf den Server hochgeladen werden. Die Funktionalität dieses Packages wird insbesondere von den Komponenten in Anspruch genommen, die die verschiedene Dokumenttypen repräsentieren.

2.3 Aufbau der Packagehierarchie

Bereits bei der Planung des Whiteboards wurde eine Unterteilung in verschiedene Bereiche vorgenommen. Die Zeichenfläche wird als ein Teil gesehen, in dem die verschiedenen Repräsentationen der Objekte auf dem Server dargestellt werden. Verschiedene Elemente, die in die grafische Benutzungsoberfläche integriert werden, sollen die Manipulation der Repräsentationen vereinfachen. Die Repräsentationen stellen einen weiteren Bereich dar. Um die Klassen, die durch ihre Funktion als zusammengehörend betrachtet werden können auch zusammengehörend zu verarbeiten, wurde die in Abbildung 2.1 dargestellte Aufteilung in Packages³ vorgenommen. Durch die Eingliederung des Whiteboard-Source-Codes in die Package-Struktur von sTeam ist der gemeinsame Zugriff auf den Source-Code über das Versionsmanagementsystem CVS⁴ gegeben.

¹z.B. `ARRANGE_OBJECT` für die Veränderung der Position innerhalb des Raumes.

²Im Falle eines `ARRANGE_OBJECT`-Events die neue Position des Objektes innerhalb des Raumes.

³Ein Package bündelt mehrere Klassen und erlaubt so die Strukturierung des Source-Codes.

⁴Concurrent Version System.

Packagename	Packageinhalt
<code>steam.modules.whiteboard</code>	Startklassen des des WhiteBoard Clients
<code>steam.modules.whiteboard.control</code>	Steuerungselemente und grafische Benutzungsoberfläche
<code>steam.modules.whiteboard.event</code>	Spezifische Events und Eventlistener des Whiteboards
<code>steam.modules.whiteboard.component</code>	Die Repräsentationen der verschiedenen Objekttypen
<code>steam.modules.whiteboard.component.msimages</code>	Bilder für die Verwendung im münsteraner Layout
<code>steam.modules.whiteboard.component.images</code>	Bilder für die Verwendung im paderborner Layout
<code>steam.modules.applicationlauncher</code>	Manager für den automatischen upload lokal geänderter Dokumente

Abbildung 2.1: Die Packagesstruktur

2.4 Das Package `steam.modules.whiteboard`

In diesem Package befinden sich die Klassen, die zum Starten des Whiteboard-Clients notwendig sind. Die Klasse `WhiteBoardClient` stellt die Verbindung zum Server her und initialisiert eine Instanz der Klasse `GlobalPreferences`⁵, die den Zugriff auf globale Eigenschaften kapselt. So wird der Benutzername, die Verbindung `SteamConnector`, ein Cache, sowie diverse andere global benötigte Funktionen und Variablen über diese Klasse zugänglich gemacht. Nach dem Verbindungsaufbau wird in `WhiteBoardClient` die Benutzungsoberfläche zusammengestellt. `WhiteBoardClient` erstellt dazu ein neues Fenster in Form eines `javax.swing.JFrame`. Die verschiedenen Elemente wie Menü, Toolbar, Zeichenfläche und Benutzerliste werden angelegt und bieten dem Benutzer so den Zugriff auf die Daten des Servers. Wird der Client beendet sorgt `WhiteBoardClient` dafür, dass die temporär angelegten Daten in einem konsistenten Zustand bis zur nächsten Anmeldung verbleiben.

Um eine private Zeichenfläche zur Verfügung zu stellen und diese komfortabel zugänglich zu machen, wurde die Klasse `WhiteBoardPanel` implementiert. In ihr ist die Zeichenfläche enthalten, die den Inhalt des aktuellen Raumes darstellt, sowie eine Zeichenfläche, die es ermöglicht, Dinge in einem privaten Arbeitsbereich vorzubereiten. `WhiteBoardPanel` übernimmt die Verteilung der Benutzereingriffe an die im Vordergrund liegende Zeichenfläche. Operationen wie das Erstellen eines neuen Raumes werden von den verschiedenen Oberflächenelementen (Toolbar, Menü) an diese Klasse delegiert. Das Umschalten zwischen aktuellem Raum, dem privaten Arbeitsbereich und des Container-Browsers⁶ teilt den Inhalt eines Containers auf einer separaten Zeichenfläche dar. kann so einfach per Mausklick erfolgen. Die einzig notwendige Änderung besteht darin, dass die private Zeichenfläche überlappend dargestellt wird und Befehle, wie das Erzeugen neuer Objekte, Farbänderungen, etc. an die private Zeichenfläche übermittelt werden. Durch die Zusammenfassung der Steuerung der Zeichenflächen über diese Klasse ist eine Erweiterbarkeit gewährleistet, die es ermöglicht, in Zukunft weitere Zeichenflächen ohne großen Programmieraufwand zur Verfügung zu stellen, um so eventuell das parallele Arbeiten in mehreren Räumen zu gestatten. So wurde z.B. der Container Browser hinzugefügt, der den Inhalt eines Containers in einer separaten Zeichenfläche darstellt. Da die Steuerung über das `WhiteBoardPanel` gebündelt erfolgt, kann auf die speziellen Eigenschaften des Container Browsers (Zeichnungselemente wie Linie Kreis und Rechteck sind innerhalb eines Containers nicht erlaubt) eingegangen werden, und so werden die entsprechenden Oberflächenelemente beim aktivieren des container Browsers deaktiviert und stehen somit während der Verwendung des Container Browsers nicht zur Verfügung.

⁵Enthält Methoden und Variablen, die Packageunabhängig von allen Klassen benötigt werden.

⁶S

2.5 Das Package `steam.modules.whiteboard.control`

Das Package `steam.modules.whiteboard.control` fasst die grafischen Steuerungselemente des Whiteboard-Clients zusammen. Mit Hilfe dieser Steuerungselemente können die dargestellten Repräsentationen schnell und einfach manipuliert werden. Abhängig vom Typ der Repräsentationen werden die Farb- und Schriftänderungen zugänglich gemacht. Die Toolbar bietet über verschiedene Buttons den Zugriff auf generelle Funktionen wie das Erzeugen und Löschen der Repräsentationen. Die Benutzerliste stellt die im aktuellen Raum befindlichen Benutzer dar und gibt Auskunft über deren aktuellen Status⁷. Sie erlaubt ferner die Veränderung des eigenen Status. In der Klasse `GlobalPreferences` werden Funktionen und Variablen gekapselt, auf die von allen Klassen des Whiteboard-Clients zugegriffen wird. Die Optimierung des Netzverkehrs über einen integrierten Cache bei der Abfrage von Dokumentdaten und Bildern ist genauso realisiert wie die Speicherung global gültiger Variablen wie der Benutzername, das Benutzerobjekt und die `SteamConnector`-Instanz, die die Verbindung zum Server darstellt. So können die verschiedenen Klassen über eine zentrale Stelle auf diese Eigenschaften zugreifen, so dass Änderungen nur an einer Stelle gemacht werden müssen.

2.5.1 Die Steuerungselemente

Die *Benutzerliste* wird durch die Klasse `AwarenessControl` implementiert. Die Darstellung der einzelnen Benutzer erfolgt als vertikale Liste. Die `AwarenessControl` benutzt zur Darstellung der einzelnen Benutzer Instanzen der Klasse `UserProxyComponent`. Ein `UserProxyComponent` wird von der zentralen Klasse `steam.modules.whiteboard.component.WhiteBoard` für jeden im Raum befindlichen Benutzer der Benutzerliste hinzugefügt und fungiert als Repräsentation des entsprechenden Benutzers in der Benutzerliste. Der `UserProxyComponent` stellt durch die Überlagerung der `paint()`-Methode den Benutzer durch sein auf dem Server abgelegtes Bild und seinen Benutzernamen dar. Eine Statusanzeige gibt Auskunft über die Erreichbarkeit des Benutzers. Zu diesem Zweck registriert sich der `UserProxyComponent` als Event-Empfänger bei dem Benutzer-Objekt auf dem Server. Jede Veränderung des Benutzer-Status wird der Instanz somit direkt mitgeteilt, der Status ist immer auf dem aktuellen Stand. Der `UserProxyComponent` stellt dem Benutzer ein Popup-Menü zur Verfügung, mit dem eine EMail an die entsprechende Person versandt werden kann. Die EMail Adresse wird aus den Benutzerdaten ausgelesen und braucht nicht extra eingegeben zu werden. Für den lokalen Benutzer kann über dieses Menü zusätzlich der Status verändert werden. Damit ist es möglich, anderen die Abwesenheit zu signalisieren, wenn man z.B. kurz nicht am Rechner sitzt, aber den Whiteboard-Client nicht beenden möchte. Über das Menü lässt sich auch das Benutzer-Bild verändern. Diese Änderung erfolgt

⁷Die Benutzer können als Anwesend, Abwesend, Beschäftigt oder Offline markiert sein.

mit Hilfe eines Web-Browsers, der dann automatisch mit der entsprechenden Seite, die die Änderung ermöglicht, gestartet wird. Zur Darstellung der Benutzerliste wird eine Instanz der Klasse `AwarenessWrapperPanel` erzeugt. Diese enthält die `AwarenessControl` und sorgt für die korrekte grafische Darstellung, unabhängig von dem `LayoutManager`, der im übergeordneten Panel verwendet wird.

Der speziell für den Einsatz im `WhiteBoardClient` entwickelte *Farbwähler* ist durch die Klasse `ColorPanel` realisiert. Zur Farbauswahl stellt Java einen Standard-Farbwähler bereit, der aber für die schnelle Änderung von Farben zu komplex ist. Deshalb wurde der spezielle Farbwähler implementiert, der die Farbänderung durch den Benutzer mit wesentlich weniger Aufwand ermöglicht. `ColorPanel` stellt sechs verschiedenfarbige Farbknöpfe zur Verfügung, über die eine Farbänderung vorgenommen werden kann. Diese werden horizontal nebeneinander angeordnet und bieten so schnellen Zugriff auf die einzelnen Farben. Die Farbknöpfe werden durch `CComponent` dargestellt und stellen je eine Farbe dar. Bei der Anwahl eines Farbknopfes wird die Farbänderung an die zentrale Klasse `steam.modules.whiteboard.component.WhiteBoard` weitergegeben. Diese setzt die neue Farbe bei den selektierten Repräsentationen. `CComponent` ist von `javax.swing.JComponent` abgeleitet und bietet durch die Überlagerung der Mausbehandlungs-Methoden die Möglichkeit, per Doppelklick einen Standard-Farbwähler, der von Java bereitgestellt wird, zu öffnen, um die Farbe des Farbknopfes den eigenen Bedürfnissen anzupassen.

Einige Repräsentationen stellen Text dar. Um die Änderung der Schriftart, der Schriftgröße und des Schriftschnitts dieser Repräsentationen zu ermöglichen, wurde mit der Klasse `FontPanel` ein *Schriftwähler* implementiert. `FontPanel` liest alle auf dem System verfügbaren Schriftarten ein und fügt diese in eine `javax.swing.JComboBox`. Die wählbaren Schriftgrößen 10, 12, 14, 18, 24, 36 und 72 werden über eine weitere `JComboBox` zugänglich gemacht. Die verschiedenen Schriftschnitte normal, kursiv, fett und fett/kursiv sind in einer weiteren `JComboBox` zusammengefasst. Das `FontPanel` stellt die drei Ausprägungen der `JComboBox` horizontal nebeneinander dar. So ist der direkte Zugriff auf alle Schriftoptionen gewährleistet. Das `FontPanel` stellt die aktuell gesetzte Schrift der gewählten Repräsentation dar. Bei einer Änderung durch den Benutzer wird die neu ausgewählte Schrift an die zentrale Klasse `steam.modules.whiteboard.component.WhiteBoard` übermittelt. Diese setzt die neue Schrift als neuen Standard und stellt diese auf den selektierten Repräsentationen ein.

Die Klasse `WBToolBar` implementiert die *Symbolleiste* des Whiteboard-Client. Um alle Funktionen zugänglich zu machen, enthält `WBToolBar` zwei Instanzen von `javax.swing.JToolBar`, die wiederum über entsprechende Bedienelemente den Zugriff auf die Funktionen des Whiteboards bieten. `WBToolBar` ist von `java.awt.JPanel` abgeleitet und stellt die beiden Toolbars untereinander dar. `WBToolBar` generiert Buttons für den Zugriff auf die Funktionen und fügt sie den innenliegenden Toolbars hinzu. Die Steuerung und Verarbeitung der Button-Events erfolgt nicht durch

die innenliegenden Toolbars, sondern durch `WBToolBar` selbst. Neben den Buttons, mit denen verschiedene neue Objekte erstellt, der Chat geöffnet, ein Telepointer aktiviert, die private Zeichenfläche geöffnet und Objekte gelöscht werden können, werden auch der Schrift- und Farbwähler zu diesen Toolbars hinzugefügt. So kann auf alle wichtigen Funktionen direkt zugegriffen werden.

2.5.2 Kapselung globaler Variablen und Methoden

Die Klasse `GlobalPreferences` stellt diverse statische Methoden und Variablen zur Verfügung, die zu Beginn der Sitzung initialisiert werden und von da an von allen Klassen des Whiteboard-Clients abgefragt werden können. Durch die Einrichtung dieser zentralen Instanz, die die von den unterschiedlichsten Klassen benötigten Funktionen kapselt, ist es möglich, verschiedenartige Optimierungen beim Zugriff auf die benötigten Daten zu implementieren. Die wichtigsten Funktionen und Optimierungen werden im folgenden dargestellt.

Cache: Zur Darstellung einiger Objekte laden die Repräsentationen die entsprechend auf dem Server für diesen Typ gesetzten Icons. Eine kleine Tür z.B. stellt einen Raum dar. Damit nicht jede Repräsentation eines Raumes eine neue Anfrage an den Server richten muss, um die Daten für dieses Icon zu laden, lädt `GlobalPreferences` zu Beginn einer Sitzung die Icons für die Standard-sTeam-Objekte wie Room, Container und Document vom Server und speichert diese im Cache. Fordern die Repräsentationen nun das entsprechende Icon an, kann `GlobalPreferences` auf den Cache zurückgreifen, die Repräsentation kann ohne Serverzugriff initialisiert werden. Im Cache werden die Dokumentdaten anhand ihrer eindeutigen Objekt-ID abgelegt. Für die gesetzten Icons reicht die ID allein aus, da bei einer Änderung des Icons das entsprechende Attribut des Objektes geändert wird und anschließend auf ein neues Objekt mit einer unterschiedlichen ID verwiesen wird. Für die Daten eines Document-Objektes hingegen reicht es nicht aus, die Daten nur anhand ihrer Objekt-ID zu differenzieren, da sie nach dem Anlegen des Dokumentes noch geändert werden können. Daher wird für den Inhalt von Dokumenten die Zeit, zu der das Dokument zuletzt modifiziert wurde, zusammen mit der ID als Schlüssel abgespeichert. Bei der Anforderung eines Dokumenteninhaltes kann so vor dem Beginn des Datentransfers vom Server abgefragt werden, ob die aktuelle Version bereits im Cache vorliegt. Der Cache ist zweistufig aufgebaut. Während einer Sitzung bereits angeforderte Daten sind in einer `java.util.HashMap`⁸ abgelegt, die als Schlüssel die Objekt-ID, bzw. Objekt-ID in Kombination mit dem Datum der letzten Änderung, und als Wert die entsprechenden Dokumentdaten enthält. Daten, die während vorausgegangener Sitzungen schon abgefragt wurden, sind in einem Verzeichnis auf der Festplatte abgelegt. Als Dateiname dient die entsprechende Objekt-ID, bzw. die Objekt-ID in Kombination

⁸Enthält (Schlüssel,Wert)-Paare, die Werte können mit Hilfe des Schlüssels effizient abgefragt werden. Schlüssel sind eindeutig.

mit dem Datum der letzten Änderung. Bei der Anforderung von Dokumentendaten sucht `GlobalPreferences` erst in der im Speicher liegenden `HashMap`. Sind die angeforderten Daten dort nicht zu finden, wird versucht diese aus dem Verzeichnis auf der Festplatte zu laden. Erst wenn die Daten auch dort nicht zu finden sind, lädt `GlobalPreferences` den Dokumenteninhalte vom Server. Die Klassen, die die Dokumentendaten anfordern, sind so von der Art der Datenbeschaffung entkoppelt.

Cut, Copy, Paste : Mit dem Whiteboard-Client soll die Möglichkeit gegeben werden, Objekte in andere Räume zu bewegen, Kopien anzulegen und Objekte von der privaten Zeichenfläche in öffentlich zugängliche Räume zu transferieren. Dieses ist im Whiteboard-Client über die Funktionen Cut, Copy und Paste möglich. Die Klasse `GlobalPreferences` verwaltet die Objekte, die per Cut ausgeschnitten oder mit Copy zum Kopieren markiert worden sind. `GlobalPreferences` speichert die markierten Objekte in einem `java.util.Vector`⁹. Es speichert die zum Einfügen markierten Objekte unabhängig davon, ob sie ausgeschnitten oder zum Kopieren markiert wurden. Die unterschiedliche Behandlung der Objekte bei der Einfügeoperation wird von der Klasse `steam.modules.whiteboard.component.WhiteBoard` vorgenommen. Diese überprüft ein in den Repräsentationen gesetztes Flag und entscheidet so, ob das Objekt ausgeschnitten wurde und in den neuen Raum bewegt wird, oder ob eine Kopie angelegt werden soll. Im letzten Fall wird eine Kopie des markierten Objektes in dem Raum angelegt und das markierte Objekt verbleibt in seinem ursprünglichen Raum. Ein Problem tritt auf, wenn ein Benutzer ein Objekt löscht, welches von einem anderen Benutzer zuvor zum Einfügen markiert wurde. `GlobalPreferences` registriert sich daher als Empfänger des Lösch-Events bei den markierten Objekten. Werden diese nun gelöscht, erhält der Benutzer eine Rückmeldung darüber, dass die Einfügeoperation nicht mehr möglich ist. Eine weitere Möglichkeit, diesen Konflikt zu lösen, hätte darin bestanden, die markierten Objekte bereits bei der Markierungsoperation zu kopieren, um sie unabhängig von späteren Löschoptionen anderer Benutzer verfügbar zu haben. Diese Variante wurde verworfen, da sie für jede Markierungsoperation Netzwerkverkehr erzeugt hätte und durch die Kopieroperationen auf dem Server, die in den meisten Fällen als unnötig anzusehen gewesen wären, eine Vielzahl von temporären Objekten auf dem Server anzulegen gewesen wären. Aus diesem Grunde wurde auch nach Rücksprache mit den Entwicklern des Server-Kerns der oben geschilderte Ansatz präferiert.

Download: Für das Laden eines Dokumentes vom Server stellt die Klasse `GlobalPreferences` zwei Methoden zur Verfügung. Mit `getContentOf()`, die als Parameter eine Instanz von `SteamObject` verlangt, wird der Dokumenteninhalte im Vordergrund geladen. Mit der weiteren Ausführung des Whiteboard-Clients wird gewartet, bis die Dokumentendaten vollständig vom Server geladen wurden. Mit dieser Methode werden z.B. die Icons für die Repräsentationen der Standard-Steam-Objekte geladen. Diese Icons haben eine Größe von nicht mehr als einem Kilobyte. Ohne diese vollständig geladen zu haben, ist eine weitere Ausführung

⁹Implementiert eine lineare Liste mit Einfüge-, Löschoptionen und Suchoperation.

des Programms augenscheinlich nicht sinnvoll, da die entsprechenden Repräsentationen ohne die Icon-Daten nicht dargestellt werden können. Um größere Dokumente von Server zu laden stellt `GlobalPreferences` die Methode `download()` zur Verfügung. Diese lädt die Dokumentendaten im Hintergrund vom Server. Während des Downloads der Dokumentdaten kann mit dem Whiteboard weitergearbeitet werden. Werden Daten über die Methode `download()` geladen, wird zuvor der Cache abgefragt, ob die aktuelle Version des Dokumentes bereits lokal gespeichert ist. Ist dieses nicht der Fall, wird die Anfrage an den Server weitergeleitet. Es ist vorgesehen, dass die Instanz, die den Download initiiert, dem Benutzer eine Rückmeldung über den Fortschritt des Downloads gibt. Zu diesem Zweck wurde die Klasse `DownloadObserver` erstellt. Der Download der Daten erfolgt innerhalb der Klasse `SteamConnector`. Wird der Methode `download()` als Parameter eine `DownloadObserver`-Instanz übergeben, so wird diese an den `SteamConnector` weitergegeben. Dieser benachrichtigt den `DownloadObserver` über den Fortschritt des Downloads, indem er nach jedem erfolgreich geladenem Datenpaket die Methode `indicate()` des `DownloadObserver` aufruft. Innerhalb dieser Methode reagiert der `DownloadObserver` auf den Fortschritt des Downloads und aktualisiert den Zustandsbalken, der zur Fortschrittsanzeige verwendet wird.

Mitteilungs- und Eingabedialoge: Um den Benutzer über bestimmte Ereignisse zu informieren oder Informationen abzufragen, z.B. unter welchem Namen ein neuer Raum angelegt werden soll, ist es notwendig, dem Benutzer die Mitteilung oder die Eingabe in kleinen Dialogen zu präsentieren bzw. abzufragen. `GlobalPreferences` bietet die Möglichkeit, mittels verschiedener Methoden Mitteilungen an den Benutzer auszugeben oder eine Eingabe abzuwarten. Durch diese Kapselung ist es möglich, die diversen Dialoge an einer Stelle im Source-Code zu ändern um die neue Funktionalität für den gesamten Client zur Verfügung zu stellen. Der Aufbau der Dialoge orientiert sich an den von Keil Slawik entwickelten Kriterien zur Reduzierung erzwungener Sequentialität.

Bedienelementeverwaltung: Um den Status der Bedienelemente der Toolbar, des Menüs und der Popup-Menüs der einzelnen Repräsentationen zu synchronisieren, werden diese Bedienelemente in zwei `HashMap`-Instanzen gespeichert. In einer `HashMap` werden die Menüeinträge abgelegt, in der anderen werden die Buttons aus der Toolbar gespeichert. Als Schlüssel dienen vordefinierte Konstanten. Eine solche Konstante lautet z.B. `MENUITEM_TELEPOINTER`. Mit den Methoden `registerMenuItem()` und `registerMenuButton()` werden die Bedienelemente in die Maps aufgenommen. Um den Telepointer zu aktivieren gibt es zwei Möglichkeiten, entweder über einen Button in der Toolbar oder über einen Eintrag im KontextMenü der Zeichenfläche. Die Toolbar und die Zeichenfläche registrieren ihre Bedienelemente zur Aktivierung des Telepointers mit den Methoden `registerMenuButton()` bzw. `registerMenuItem()`. Bei der Aktivierung des Telepointers durch den Benutzer wird durch die aktivierende Instanz die Methode `changeMenuItemSelection()` aufgerufen, die als Parameter den Schlüssel der betroffenen Bedienelemente und deren neuen Status erhält. In dem beschriebenen

Falle lauten die Parameter “`GlobalPreferences.MENUITEM_TELEPOINTER`” und “`true`”. In `changeMenuItemSelection()` wird in beiden `HashMap`-Instanzen nach entsprechenden Bedienelementen gesucht und ihr Status auf “selektiert” gesetzt. In gleicher Weise ist es mittels der Methode `changeMenuItemState()` möglich, bestimmte Bedienelemente zu deaktivieren. Während die private Zeichenfläche angezeigt wird soll z.B. der Telepointer nicht aktiviert werden können. Durch den Aufruf von `changeMenuItemState(MENUITEM_TELEPOINTER, false)` werden die Bedienelemente zur Aktivierung des Telepointers deaktiviert. Sie werden ausgegraut dargestellt und signalisieren dem Benutzer somit eindeutig, dass diese Funktion zur Zeit nicht zur Verfügung steht. Dies wird u.a. verwendet, wenn der Container Browser aktiv ist um so die Oberflächenelemente, die zur Erzeugung der Elemente Linie, Kreis, Rechteck etc. verwendet werden können zu deaktivieren.

Weitere Variablen und Methoden: In der Klasse `GlobalPreferences` sind die für alle Klassen des Whiteboard-Clients interessanten Variablen und Methoden gekapselt. Auf einige wurde bereits eingegangen, desweiteren enthält `GlobalPreferences` einen `Cursorcache`. Die Instanzen der Repräsentationen holen sich aus diesem Cache eine Referenz auf eine `java.awt.Cursor`-Instanz, um die entsprechende Cursorform für die Darstellung der Repräsentation verfügbar zu haben. Gerade bei einem Raumwechsel werden unter Umständen sehr viele Repräsentationen zur Entfernung aus dem Speicher freigegeben und viele neue Repräsentationen erzeugt. Durch eine zentrale Stelle zur Verwaltung der verschiedenen Cursorformen müssen die einzelnen Repräsentationen nicht einzeln die verschiedenen `Cursor`-Objekte anlegen, sondern es reicht, eine entsprechende Referenz auf einen `Cursor` zu speichern. Diese Referenz holen sich die Klassen aus dem `Cursorcache`. Änderungen der Cursorformen können so zentral vorgenommen werden und wirken sich auf den gesamten Client aus. Ferner bietet `GlobalPreferences` den Zugriff auf verschiedene globale Variablen, z.B. ob Antialiasing¹⁰ beim Zeichnen der Repräsentationen verwendet werden soll, eine Instanz eines `javax.swing.JColorChooser`, die Instanz des `SteamConnector`, über den der Datenaustausch mit dem Server stattfindet, einen `javax.swing.JFileChooser`-Dateiauswahldialog, Adresse und Port des Web-Servers, der Zugriff auf die Daten des sTeam-Servers bietet, eine Referenz auf das Fenster, in dem der Whiteboard-Client dargestellt wird, die `SteamUser`-Instanz, die den lokalen Benutzer darstellt, den Benutzernamen und das Verzeichnis, in dem sich der Festplattencache befindet .

2.6 Das Package `steam.modules.whiteboard.event`

In diesem Package sind Klassen definiert, die den Datenaustausch über selbst definierte Events innerhalb des Whiteboards ermöglichen. Für das Whiteboard-

¹⁰Verfahren, das bei Darstellung von Buchstaben und Grafikobjekten auf dem Monitor die unschönen, treppenartigen Kanten glättet. Dies erfolgt durch eine Berechnung von Farbverläufen zwischen der Objektfarbe und der Hintergrundfarbe.

interne Eventsystem wurden zwei Events definiert, `SelectionEvent` und `RoomChangeEvent`. Die Klasse `SelectionEvent` steht für das Ereignis der Selektion einer Repräsentation. Bei der Selektion erzeugt die selektierte Repräsentation eine neue Instanz von `SelectionEvent`, und enthält die Information, ob während der Selektion die Control-Taste gedrückt wurde. Klassen, die sich bei der Repräsentation als Empfänger eines solchen Events registriert haben, werden so über die Selektion benachrichtigt und erhalten zugleich die Information, ob Control während der Selektion gedrückt war. Empfänger eines `SelectionEvent` müssen das Interface `SelectionListener` implementieren, um über dieses Ereignis unterrichtet werden zu können. Durch die Information, ob Control während der Selektion gedrückt wurde, können Empfänger dieses Events unterschiedlich auf die Selektion reagieren. In der Bedienung macht es den Unterschied, dass bei der Selektion einer Repräsentation ohne Drücken der Control-Taste die vorhergehende Selektion rückgängig gemacht und die neu selektierte Repräsentation als selektiert dargestellt wird. Wird die Control-Taste bei der Selektion gedrückt, so wird die neu selektierte Repräsentation der bereits bestehenden Selektion hinzugefügt.

Der zweite selbst definierte Event dient der Benachrichtigung über den Wechsel in einen anderen Raum. Der Event wird durch die Klasse `RoomChangeEvent` implementiert. Empfänger dieses Events implementieren das Interface `RoomChangeListener`. Dieser Event wird von der Zeichenfläche an einige Repräsentationen gesendet, so wird z.B. die Repräsentation, die eine Bild darstellt, über den Raumwechsel informiert. Falls diese Repräsentation die Bilddaten noch nicht vollständig vom Server geladen hat, und über einen Wechsel des Raumes unterrichtet wird, bricht sie den Ladevorgang ab.

2.7 Das Package `steam.modules.whiteboard`. - component

Die Klassen, die in diesem Package zusammengefasst sind, implementieren die Repräsentationen. Die Repräsentationen sind die grafischen Elemente, die auf der Zeichenfläche dargestellt werden und über verschiedene Interaktionsmöglichkeiten die Manipulation der auf dem Server gespeicherten Objekte erlauben. Der in Abbildung 2.2 dargestellte Ableitungsbaum macht deutlich, dass alle Repräsentationen Methoden von der zentralen Klasse `WhiteBoardComponent` erben. In dieser werden generelle Eigenschaften wie die Möglichkeit, Repräsentationen mit der Maus zu verschieben, Bereitstellung eines Kontextmenüs, Verarbeitung von Events und andere Funktionen implementiert, die von allen Repräsentationen benötigt werden. Einige direkt von `WhiteBoardComponent` abgeleitete Klassen stellen speziellere Funktionalitäten für die von ihr abgeleiteten Klassen zur Verfügung. `GraphicComponent` macht die Repräsentationen z.B. größenveränderbar. An den Blättern des Ableitungsbaumes finden sich die Klassen, die die konkreten Ausprägungen der Repräsen-

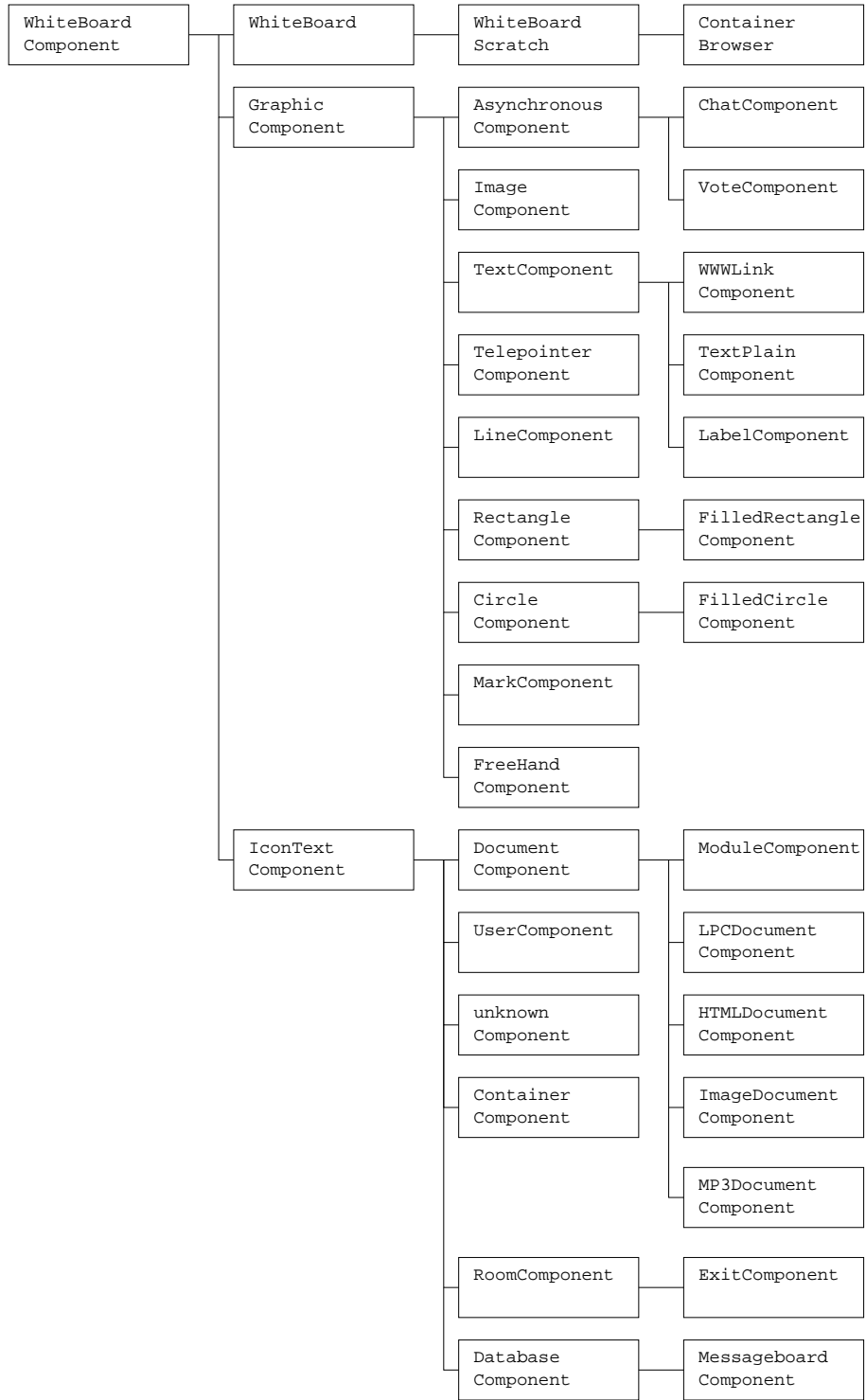


Abbildung 2.2: Die Klassenhierarchie

tationen für die auf dem Server gespeicherten Objekte implementieren und Funktionalität bereitstellen, mit der die spezifischen Eigenschaften des repräsentierten Objektes manipuliert werden können. In Abbildung 2.3 wird ein Überblick über die Funktion der einzelnen Klassen gegeben. Die einzelnen Klassen werden später erläutert.

Klassenübergreifende Konzepte

Die einzelnen Klassen implementieren die verschiedenartigen Repräsentationen. Sie sind als eigenständige Komponenten zu betrachten, die über einen durch ihre Position und Größe definierten Ausgabebereich verfügen, in den die Komponente zeichnen kann. Die Position der Komponente wird relativ zu der umgebenen Komponente angegeben. Die Klasse `WhiteBoard` implementiert die Zeichenfläche, zu dieser werden die einzelnen Komponenten hinzugefügt und erhalten so Ausgabebereiche innerhalb der Zeichenfläche. Die Klasse `WhiteBoard` übernimmt dabei diverse Steuerungsaufgaben und verwaltet die dargestellten Komponenten. `WhiteBoard` stellt verschiedene Methoden zum lokalen Erzeugen neuer Objekte zur Verfügung und sorgt dafür, dass für die Objekte, die in dem dargestellten Raum existieren, eine lokale Komponente als Repräsentation vorhanden ist. Die Interaktionen, die an den einzelnen Komponenten vorgenommen werden können, werden von den Komponenten selbst implementiert. Positions- und Größenveränderungen werden unabhängig von der `WhiteBoard`-Instanz vorgenommen. Da die Klasse `WhiteBoardComponent` von `javax.swing.JComponent`¹¹ abgeleitet ist, ist es durch einfaches Überschreiben der entsprechenden Methoden möglich, die Verarbeitung der Mausaktionen selbst zu übernehmen. So werden von den verschiedenen Komponenten spezielle Bereiche definiert, in denen der Benutzer die Komponenten mit der Maus verändern kann. Durch die Veränderung des Mauszeigers wird dem Benutzer eine Rückmeldung über die gerade mögliche Interaktion gegeben.

Für jedes Objekt, das sich in dem vom `WhiteBoard` dargestellten Raum befindet, wird eine Instanz der Klasse erzeugt, die als Repräsentation des entsprechenden Objekttyps dient. Diese wird zum `WhiteBoard`-Objekt hinzugefügt und so auf der Zeichenfläche dargestellt. In jedem der dargestellten Komponenten wird eine Referenz auf das entsprechende Server-Objekt gespeichert. Um die lokalen Eigenschaften der Komponenten mit den Attributen des Server-Objektes zu synchronisieren, werden die Komponenten bei einer Änderung der Attribute auf dem Server benachrichtigt und können ihre Eigenschaften aktualisieren. Genauso werden die Manipulationen, die der Benutzer lokal an den Komponenten vornimmt, auf den Server übertragen. So werden andere Clients, die als Empfänger von Änderungen der entsprechenden Objekten registriert sind, durch den Event-Mechanismus von den Änderungen in

¹¹`JComponent` ist die Basisklasse für die grafischen Bedienelemente, die das Swing-Paket zur Verfügung stellt. `JComponent` stellt bereits Methoden zur Positions- und Größenveränderung bereit. Ebenso ist es möglich, zu einem `JComponent` verschiedene Komponenten hinzuzufügen. Diese werden innerhalb des Ausgabebereiches von `JComponent` dargestellt.

Klassenname	Funktion der Klasse
WhiteBoardComponent	Oberklasse, erlaubt Mausinteraktionen
WhiteBoard	Zeichenfläche
WhiteBoardScratch	private Zeichenfläche
ContainerBrowser	stellt Containerinhalt dar
IconTextComponent	Oberklasse für Repräsentationen der Standardobjekte von sTeam
ContainerComponent	Repräsentation für das Container Objekt
RoomComponent	Repräsentation für das Room Objekt
ExitComponent	Repräsentation für das Exit Objekt
ModuleComponent	Repräsentation für das Module Objekt
UserComponent	Repräsentation für das User Objekt
unknownComponent	Stellt Objekte dar, für die keine spezielle Repräsentation existiert
DatabaseComponent	Repräsentation eines Datenbank Objektes
MessageBoardComponent	Repräsentation eines Message- Boards
DocumentComponent	Repräsentation für das Document Objekt
LPCDocumentComponent	Repräsentation für ein LPC Document
HTMLDocumentComponent	Repräsentation für ein Html Document
ImageDocumentComponent	Repräsentation für ein Bild
MP3DocumentComponent	Repräsentation für ein MP3 Document
GraphicComponent	Oberklasse der grafischen Objekte, erlaubt Größenänderung
FreeHandComponent	kleine "Frei Hand" Zeichnung
LineComponent	Linie
CircleComponent	Kreis
FilledCircleComponent	ausgemalter Kreis
RectangleComponent	Rechteck
FilledRectangleComponent	ausgemaltes Rechteck
MarkComponent	Markierung
ImageComponent	Bild
TelepointerComponent	Telepointer
TextComponent	Erlaubt das Setzen einer Schriftart
LabelComponent	einzeiliger Text
TextPlainComponent	mehrzeiliger Text
WWWLinkComponent	Link ins Internet
AsynchronousComponent	Oberklasse für Komponenten, die sich zeitweise asynchron Verhalten sollen
ChatComponent	Chat
VoteComponent	Abstimmungstool

Abbildung 2.3: Die Funktionen der einzelnen Klassen

Kenntnis gesetzt.

Jede Komponente bietet die Möglichkeit, auf spezielle Funktionen über ein Kontextmenü zuzugreifen. Über dieses Menü sind die Funktionen zugreifbar, die nicht unmittelbar über eine Interaktion am Komponent selbst umgesetzt werden können. Für alle Komponenten verfügbare Funktionen, wie z.B. löschen, cut, copy und paste, werden dem Kontextmenü in `WhiteBoardComponent.initContextMenu()` hinzugefügt. Die Subklassen überschreiben diese Methode und fügen ihre komponentenspezifischen Einträge ein. Wählt der Benutzer einen Eintrag aus, so wird dieser ausgewählte Eintrag an die Methode `performCommand()` weitergereicht. Diese führt die gewählte Funktion aus. Die Funktionen, die in `WhiteBoardComponent.initContextMenu()` dem Kontextmenü hinzugefügt wurden, werden in der Methode `WhiteBoardComponent.performCommand()` verarbeitet. Die einzelnen Komponenten implementieren die Funktionalität, indem sie die `performCommand()`-Methode überschreiben und dort die Komponentenspezifischen Funktionen einsetzen.

Der Ein- und Ausgabebereich der Komponenten wird durch die Position und die Größe definiert und ist somit rechteckig. Der Zugriff auf eine Komponente durch den Benutzer erfolgt mittels Maus. Wird diese über den Komponent bewegt, bietet dieser verschiedene Interaktionsmöglichkeiten. Da es gewollt ist, dass sich einige der Komponenten überlagern, ist diese rechteckige Erscheinungsform ein erhebliches Handicap bei der Verarbeitung der Mausektionen. Für zwei sich überlagernde Komponenten kann der unten liegende Komponent nicht mit der Maus angefasst werden. Da viele der Komponenten von sich aus nicht rechteckig sind, z.B. Linie und Kreis, wird der Bereich, in dem der Komponent auf Mausektionen reagiert, an das zu zeichnende Objekt angepasst. Zu diesem Zweck wird die Methode `contains()` aus `JComponent` überlagert. Diese wird von Java aufgerufen, um anhand der Mausposition festzustellen, ob die Maus sich zur Zeit über der Komponente befindet. Durch entsprechende Berechnungen wird dieser Bereich an die darzustellende Form angepasst und ist zudem noch als abhängig davon, ob die Komponente selektiert ist oder nicht. In dem Fall wird auch das Überfahren der Interaktionsknöpfeß.B. für die Größenänderung von der Komponente registriert.

Die Klasse `WhiteBoardComponent`

In `WhiteBoardComponent` wird die Referenz auf das von der Komponente repräsentierte Server-Objekt gespeichert. Die Subklassen können so direkt auf die `SteamObject`-Referenz zugreifen und über die Funktionen, die in dem sTeam API zur Verfügung gestellt werden, das auf dem Server liegende Objekt modifizieren. Für den Zugriff stellt auch `WhiteBoardComponent` einige Funktionen zur Verfügung, die von den Subklassen für die Modifizierung des Sever-Objekts benutzt werden. Für einige Operationen, zu denen das Setzen der Attribute auf dem Server gehört,

wird der Aufruf in einem try-catch-Block¹² verlangt, um eventuell auftretende Fehler abzufangen. `WhiteBoardComponent` kapselt diese Funktionalität in der Methode `setAttribute()` und definiert an dieser Stelle den try-catch-Block. So können die Subklassen die Attribute auf dem Server setzen, ohne für jeden Zugriff einen eigenen try-catch-Block implementieren zu müssen. Auf eventuelle Fehler beim Zugriff auf den Server kann somit an einer zentralen Stelle reagiert werden.

Die Kontrolle über Mausektionen erlangt `WhiteBoardComponent`, indem in dieser Klasse die Methode `processMouseEvent()` aus `JComponent` überschrieben wird. Diese Methode wird Java-intern aufgerufen und erhält als Parameter ein `java.awt.event.MouseEvent`-Objekt, in dem u.a. Informationen über Typ, Position des Auftretens und die gedrückte Maustaste gespeichert sind. Anhand des Typs wird die weitere Verarbeitung der Mausektionen an die Methoden `processMousePressed()`, `processMouseReleased()`, `processMouseMove()`, `processMouseClicked()` und `processMouseDragged()` delegiert. Diese erhalten als Parameter das `MouseEvent`-Objekt. In `WhiteBoardComponent.processMousePressed()` wird beim Drücken der rechten Maustaste das Kontextmenü sichtbar gemacht. Die oben genannten Methoden rufen nach ihrer Abarbeitung eine weitere Methode auf, durch deren Überschreiben den Subklassen die Möglichkeit gegeben wird, die Mausektionen des Benutzers, abhängig von der Art der Komponente, zu behandeln. Grundsätzlich wird anhand der aktuellen Mausposition ein interner Status der Komponente gesetzt, der angibt, welche Interaktion zur Zeit möglich ist. Die mögliche Interaktion wird von den Subklassen durch das Überschreiben der Methode `mouseMoved()` gesetzt. Durch diese Vorgehensweise definieren die Komponenten die Bereiche, in denen bestimmte Interaktionen möglich sind. In `WhiteBoardComponent` wird dieser Wert standardmäßig auf "Verschieben" gesetzt. Durch das Ziehen an der Komponente mit der Maus kann der Benutzer dann die Komponente neu positionieren. Durch eine Änderung des Mausursors wird dem Benutzer eine Rückmeldung über diese Möglichkeit gegeben. Die Methode `mouseDragged()`, die von `processMouseDragged()` aufgerufen wird, passt die Position der Komponente an die im `MouseEvent` gespeicherte Mausposition an. Nach Abschluss der Interaktion werden die Attribute des Server-Objektes aktualisiert. Dies geschieht in der Methode `mouseReleased()`. Sie erkennt anhand des zuvor gesetzten Wertes für die mögliche Interaktion, welche Veränderungen an der Komponente gemacht worden sind und aktualisiert die entsprechenden Attribute des Server-Objektes.

2.7.1 Die Zeichenflächen

Der Whiteboard-Client stellt zwei unterschiedliche Zeichenflächen zur Verfügung. Die durch `WhiteBoard` implementierte Zeichenfläche stellt den Inhalt des Raumes dar, in dem sich der Benutzer gerade aufhält. Die Subklasse `WhiteBoardScratch`

¹²Ein solcher Block wird in Java verwendet, um Laufzeitfehler abzufangen.

implementiert die private Zeichenfläche, die es den Benutzern erlaubt, unabhängig von dem gerade besuchten Raum für andere nicht sichtbare Objekte darzustellen.

Die Klasse `WhiteBoard`

In `WhiteBoard` ist die Zeichenfläche implementiert. Die in `WhiteBoardComponent` zur Verfügung gestellte Referenz auf das entsprechende Server-Objekt verweist in `WhiteBoardComponent` auf den Raum, dessen Inhalt auf der Zeichenfläche dargestellt wird. Diese Klasse übernimmt die Verwaltung der Komponenten, die auf ihr dargestellt werden und die die Objekte auf dem Server repräsentieren. Werden Objekte aus dem dargestellten Raum entfernt, so erhält `WhiteBoard` einen entsprechenden Event und entfernt die Komponente, die dieses Objekt dargestellt hat. Werden dem Raum neue Objekte hinzugefügt, so erstellt `WhiteBoard` abhängig vom Typ des neuen Objektes eine neue Komponente, durch die das Objekt auf der Zeichenfläche dargestellt wird. `WhiteBoard` registriert sich bei allen Komponenten als Empfänger eines `SelectionEvent`. Es wird so von den Komponenten benachrichtigt, wenn diese vom Benutzer selektiert wurden. Die selektierten Elemente werden in einem `java.util.Vector` gespeichert. Sind mehrere Elemente selektiert, so kann `WhiteBoard` die vom Benutzer ausgeführte Interaktion auf alle selektierten Elemente anwenden, so dass gemeinsames Verschieben mehrerer selektierter Elemente möglich gemacht wird. Die Mehrfachselektion wird in `WhiteBoard` durch die Implementierung einer "Gummibandlinie"¹³ vereinfacht.

Betrifft der Benutzer einen neuen Raum, wird `WhiteBoard` über einen entsprechenden Event über den Wechsel informiert. Die Komponenten werden von der Zeichenfläche entfernt und eine Liste der Objekte, die sich in dem betretenen Raum befinden, vom Server geladen. Für jedes dieser Objekte wird eine neue Instanz angelegt, die das Objekt auf der Zeichenfläche darstellt. Der Raumwechsel wird in der Methode `changeRelatedSteamObject()` vorgenommen. Als Parameter erwartet diese Methode eine Referenz auf den Raum oder Container, der von dem `WhiteBoard`-Objekt dargestellt werden soll. In `WhiteBoard` werden die betretenen Räume in einer sogenannten `History` abgelegt. Diese wird durch die interne Klasse `History` implementiert. Diese speichert die bereits besuchten Räume in einer Liste. Durch entsprechende Einträge im Kontextmenü des Whiteboards kann der Benutzer so komfortabel durch die Raumstruktur navigieren.

`WhiteBoard` aboniert die Events aller enthaltener Komponenten. Diese müssen sich nicht extra als Empfänger der Events des durch sie repräsentierten Objektes registrieren, sondern `WhiteBoard` registriert sie automatisch und nimmt die Verteilung der Events auf Client-Seite vor. Diese Aufgabe wird von der internen Klasse `WB_Event.Scheduler` übernommen. Anhand der Objekt-ID, die in dem ankommenden Event gespeichert ist, wird aus den dargestellten Komponenten derjeni-

¹³Durch Ziehen der Maus auf der Zeichenfläche öffnet sich ein Rechteck, die innenliegenden Komponenten werden beim Loslassen der Maustaste selektiert.

ge ermittelt, der das entsprechende Objekt darstellt. Der Event wird zur weiteren Verarbeitung an diesen Komponenten weitergereicht, der reagiert dann angemessen auf den entsprechenden Event. Durch die Registrierung der `WhiteBoard`-Instanz als Empfänger aller Events der dargestellten Objekte erfolgt eine Optimierung des Raumwechsels. Gerade beim Wechseln des Raumes hätte jeder neu erzeugte Komponent sich als Empfänger der Events des dargestellten Objektes registrieren müssen. Tests haben gezeigt, dass dieser zusätzliche Netzverkehr gerade beim Wechsel in Räume, die viele Objekte enthalten, maßgeblich verlangsamt. Da sich bei dieser Lösung nur die `WhiteBoard`-Instanz als Empfänger registrieren muss, ist die Anzahl der Nachrichten, die an den Server geschickt werden, um Event-Empfänger zu registrieren, unabhängig von der Anzahl der Objekte, die sich in dem darzustellenden Raum befinden.

Die Eigenschaft, in verschiedene Modi zu gehen, um bestimmte Interaktionen zu erlauben, erbt `WhiteBoard` von `WhiteBoardComponent`. Während sich die unterschiedlichen Modi in `WhiteBoardComponent` auf die möglichen Interaktionen mit dem jeweiligen Komponent selbst beziehen, werden in `WhiteBoard` neue Modi definiert, um den Inhalt des dargestellten Raumes zu manipulieren. Das Erzeugen der einfachen grafischen Objekte Linie, Kreis, Rechteck, gefüllter Kreis, gefülltes Rechteck und Markierung erfolgt über solche Modi. `WhiteBoard` unterbindet während des Erstellungsvorgangs die Mausbehandlung aller anderen Komponenten, so dass der zu erzeugende Komponent andere überlagern kann, ohne dass diese die Mausaktion interpretieren. So kann eine neue Linie intuitiv mit der Maus gemalt werden. Andere Objekte wie der einzeilige Text, mehrzeiliger Text, Raum, Container und Bild werden ohne Umstellung in einen speziellen Modi durch Aufruf der Methode `createNewComponent()` auf die Zeichenfläche gebracht. Diese erwartet als Parameter ein Integer-Wert, der den Typ der neu anzulegenden Komponente spezifiziert. Innerhalb dieser Methode wird ein entsprechendes Objekt auf dem Server erzeugt, dass durch diesen Komponent dargestellt wird.

Farbänderungen und Änderungen der Schrift durch die in der Symbolleiste dargestellten Steuerungselemente `ColorPanel` und `FontPanel` werden durch den Aufruf der Methode `setColor()` zum Farbwechsel bzw. `setFont()` zur Änderung der Schrift ausgeführt. In beiden Fällen setzt `WhiteBoard` die neue Farbe bzw. Schrift als Standard für Objekte, die später neu erzeugt werden. Die entsprechende Änderung wird an alle selektierten Elemente weitergereicht. Diese passen ihre Darstellung der neu gesetzten Farbe bzw. Schrift an und aktualisieren die entsprechenden Attribute des Server-Objektes.

Des Weiteren stellt `WhiteBoard` Methoden zur Verfügung, die beim Anlegen neuer Räume, Container und Dokumente in dem Raum eine Kollisionsauflösung bei der Namensgebung implementieren. Diese Objekte werden maßgeblich anhand ihres Namens identifiziert; die Existenz zweier Objekte mit gleichem Namen in einem Raum sollte deshalb vermieden werden. Der Whiteboard-Client macht dies, indem der vom Benutzer gewählte Name vor dem Anlegen des Objektes mit den Namen

der bereits bestehenden Objekte verglichen wird. Besteht bereits ein Objekt mit dem gewählten Namen, so wird das Objekt erzeugt, und erhält als Namen die vom Benutzer gemachte Eingabe, die durch Anhängen einer Zahl ergänzt wird. Bei Dokumenten, die üblicherweise einen Namen haben, der den Namen durch einen Punkt von dem Dateityp trennt, wird diese Ergänzung entsprechend vor dem Punkt vorgenommen, so dass beim Download der Dokumentendaten keine Dateien mit ungültigen Namen angelegt werden.

Die Klasse `WhiteBoardScratch`

`WhiteBoardScratch` stellt die private Zeichenfläche dar. Sie ist von `WhiteBoard` abgeleitet und erweitert diese um spezielle Eigenschaften der privaten Zeichenfläche. Einige Funktionen, die von `WhiteBoard` geerbt werden, werden durch überschreiben der entsprechenden Methoden nicht unterstützt. `WhiteBoardScratch` reagiert nicht wie `WhiteBoard` auf einen Raumwechsel durch den Benutzer. Ebenso wird beim Wechsel des darzustellenden Raumes das Benutzerobjekt nicht in diesen bewegt. So ist es möglich, sich durch die Raumstruktur der privaten Zeichenfläche unabhängig von dem in der Zeichenfläche dargestellte Raum zu bewegen. Da der Inhalt der privaten Zeichenfläche nicht von anderen Benutzern manipuliert werden kann, können dort Daten abgelegt werden, die in dieser Form zu einem späteren Zeitpunkt in öffentliche Räume transferiert werden sollen. Allen Komponenten, die auf der privaten Zeichenfläche dargestellt werden, wird durch Überlagerung der Methode `addImpl()` ein zusätzlicher Eintrag im Kontextmenü hinzugefügt, über den der sofortige Transfer der Komponente in den in der Zeichenfläche dargestellten Raum möglich ist.

Der Zugriff auf die Funktionen erfolgt über die Symbolleiste. Durch die Kapselung der Methodenaufrufe für `WhiteBoard`-Instanzen in der Klasse `WhiteBoardPanel` werden die entsprechenden Methoden bei der im Vordergrund dargestellten Instanz ausgeführt. Durch Überschreiben der Funktion `doubleClickPerformed()` lässt sich die private Zeichenfläche durch einen Doppelklick wieder in den Hintergrund schieben.

Die Klasse `ContainerBrowser`

Die `ContainerBrowser` Instanz ist von `WhiteBoardScratch` abgeleitet und stellt auf einer separaten Zeichenfläche den Inhalt von Containern an. Wie `WhiteBoardScratch` auch, überlagert `ContainerBrowser` die Methoden von `WhiteBoard` um so seine speziellen Eigenschaften herauszubilden. So wird bei einem Wechsel des dargestellten Inhaltes innerhalb des Container Browsers der Benutzer nicht in den entsprechenden Bereich bewegt. Sondern er bleibt vollkommen unabhängig von dem im Container Browser dargestellten Inhalt. Der Container Browser kann über einen entsprechenden Button bei Bedarf ein- und ausgeblendet wer-

den. Genau wie bei `WhiteBoardScratch` bleibt der Inhalt des `ContainerBrowser` auch dann erhalten, wenn der Benutzer sich innerhalb der Navigationsstrukturen bewegt. So ist ein späterer Zugriff auf die Daten möglich.

2.7.2 Repräsentationen für die Standard-Objekte in `sTeam`

Die Standard-Objekte `Room`, `Container`, `Document`, `Exit`, `Module` und `User` werden auf der Zeichenfläche durch ihr Icon und ihren Namen als Unterschrift des Icons dargestellt. Die Basisfunktionalität der Komponenten, die die Standard-Objekte von `sTeam` repräsentieren, wird in der Klasse `IconTextComponent` implementiert. `IconTextComponent` dient als Oberklasse für die Komponenten, die Standard-Objekte von `sTeam` repräsentieren. Im Konstruktor sorgt die Klasse dafür, dass das dem Typ entsprechende Icon geladen wird. Dazu ruft `IconTextComponent` die Methode `getIcon()` aus `GlobalPreferences` auf, so dass erst der Cache auf eine gültige Version der Bilddaten des geforderten Icons überprüft wird, bevor die Bilddaten vom Server geladen werden. Durch Überschreiben der `paint()`-Methode sorgt `IconTextComponent` dafür, dass das Icon und der Text in den Ausgabebereich der Komponente gezeichnet werden. Um bei der Selektion die Selektionsmarkierung anzupassen, wird die Methode `paintSelectionLine()` überschrieben. Während der Initialisierung wird ein `java.awt.Polygon` angelegt, welches einen Linienzug enthält, der an der äußeren Grenze des Icons und des Namens entlangführt. Ist die Komponente selektiert, so wird dieses Polygon von `paintSelection()` als Selektionsmarkierung in den Ausgabebereich gezeichnet. Um auf eine Änderung des Icons während der Sitzung mit dem `Whiteboard-Client` reagieren zu können und synchron eine Anpassung vorzunehmen, wird die Methode `updateLocalProperties()` aus `WhiteBoardComponent` überschrieben. Im Falle einer Änderung des Icons wird in der Methode die gelieferte `HashMap` auf einen neuen Icon-Eintrag abgefragt. Wird dieser gefunden, lädt `IconTextComponent` das neue Icon vom Server und aktualisiert die Darstellung der Komponente auf der Zeichenfläche.

Die Klassen `ContainerComponent`, `RoomComponent` und `ExitComponent`

Diese Klassen sind Subklassen von `IconTextComponent`. `ContainerComponent` und `RoomComponent` sind direkt von `IconTextComponent` abgeleitet, `ExitComponent` ist von `RoomComponent` abgeleitet. `ContainerComponent` fügt dem aus `WhiteBoardComponent` geerbten Kontextmenü einen neuen Eintrag "Browse Container" hinzu. Dies geschieht durch überschreiben der Methode `initContextPopupMenu()`, indem die Methode aus der Oberklasse aufgerufen wird, um die Standard-Einträge in das Kontextmenü einzufügen und anschließend der neue Eintrag hinzugefügt wird. Durch überschreiben der Methode `doubleClickPerformed()` wird "Browse Container" auch bei einem Doppelklick des Benutzers ausgeführt. Wird dieser Eintrag gewählt, so wird der darzustellen-

de Arbeitsbereich der Zeichenfläche geändert. Die Zeichenfläche stellt nun den Inhalt des entsprechenden Containers dar. Mit dem gleichen Verfahren stellt `RoomComponent` einen Kontextmenüeintrag zum Wechsel in einen neuen Raum zur Verfügung. `ExitComponent` ist von `RoomComponent` abgeleitet und unterscheidet sich nur von der Implementation des Raumwechsels von `RoomComponent`. Da ein `Exit` ein Link auf einen Raum darstellt, kann nicht das repräsentierte Server-Objekt als neuer darzustellender Arbeitsbereich gesetzt werden, sondern das darzustellende Objekt muss erst durch eine in `SteamExit` bereitgestellte Methode vom Server geladen werden.

Die Klassen `DocumentComponent` und `ModuleComponent`

`DocumentComponent` repräsentiert den Objekttyp `Document`. `ModuleComponent` ist von `DocumentComponent` abgeleitet und repräsentiert den Objekttyp `Module`, der auf Server-Seite von `Document` abgeleitet ist. Diese `Document` und `Module` sind Objekte die einen Inhalt haben, die Dokumentdaten bzw. den Inhalt des entsprechenden Moduls. Sie stellen Methoden zur Verfügung, mit denen dieser Inhalt vom Server geladen und auf die lokale Festplatte abgespeichert werden kann. Der Inhalt der Objekte wird über die Methode `download()` der Klasse `GlobalPreferences` vom Server geladen. Durch die Integration des Caches in `GlobalPreferences`, wird vor dem Laden der Daten vom Server überprüft, ob die im Cache liegende Version der Dokumentendaten mit der Version auf dem Server übereinstimmt. Erst wenn nach dieser Überprüfung feststeht, dass die Daten im Cache veraltet sind, werden die Daten vom Server geladen. `DocumentComponent` trägt einen speziellen Menüpunkt `edit` in das Kontextmenü des Komponenten ein. Bei der Auswahl dieses Menüpunktes wird der Dokumenteninhalte automatisch heruntergeladen und eine lokale Anwendung zum Bearbeiten des Dokumenteninhaltes gestartet. Wird an dem Dokument eine Veränderung in Form der Speicherung eines veränderten Dokumenteninhaltes vorgenommen, so wird diese Änderung automatisch auf den Server hochgeladen. `DocumentComponent` greift dazu auf die Funktionalität des Moduls `applicationlauncher` zurück.

Die Klassen `LPCDocumentComponent`, `MP3DocumentComponent`, `HTMLDocumentComponent` und `ImageDocumentComponent`

Bei diesen Klassen handelt es sich um spezielle Ausprägungen des `DocumentComponent`. Sie ersetzen abhängig von Ihrem Medientyp den allgemeinen Punkt `open` im Kontextmenü des Komponenten durch einen spezifischeren Eintrag. So wird z.B. bei einem `MP3DocumentComponent` der Menüpunkt `open` durch `play` ersetzt.

Die Klasse `UserComponent`

Für jeden im dargestellten Raum befindlichen Benutzer wird eine Instanz eines `UserComponent` angelegt. Die Darstellung der Benutzer erfolgt durch eine Instanz von `UserProxyComponent`, der den Benutzer in der Benutzerliste repräsentiert. `UserComponent` enthält die volle Funktionalität, um den Benutzer mit seinem Bild und Namen auf der Zeichenfläche darzustellen. Über das Kontextmenü, das von `UserComponent` um einige Einträge ergänzt und von `UserProxyComponent` angezeigt wird, kann dem entsprechenden Benutzer eine EMail geschickt werden. Dazu wird der lokale EMail-Client gestartet. Der `UserComponent`, der für den lokalen Benutzer angelegt wurde bietet zusätzlich die Möglichkeit, den Chat-Status zu setzen oder ein neues Bild des Benutzers festzulegen. In einer ersten Ausführung des Prototypen diente `UserComponent` als Repräsentation der Benutzer und wurde zusammen mit den anderen Objekten auf der Zeichenfläche dargestellt. Da die Repräsentationen der Benutzer so auch durch andere Komponenten überlagert werden konnten, oder bei größeren Räumen, deren kompletter Inhalt sich nur mit Hilfe von Scrollbalken an der Zeichenfläche erschließen ließ, gab es Probleme, die anwesenden Benutzer schnell zu erfassen. Daher wurde die Benutzerliste entwickelt, in der die Benutzer durch eine Instanz von `UserProxyComponent` dargestellt werden und in einer Liste für den Benutzer schnell zu finden sind. In `UserProxyComponent` selbst sind nur die Funktionen zur korrekten Darstellung in der Benutzerliste implementiert, alle weiteren Funktionen werden durch Methodenaufrufe der entsprechenden Instanz eines `UserComponent` zur Verfügung gestellt.

2.7.3 Die grafischen Objekte

Die Klasse `GraphicComponent` bildet die Basis für die grafischen Objekte. Sie implementiert u.a. die Möglichkeit, die Größe der grafischen Objekte durch Interaktion mit der Maus zu verändern. Die Subklassen implementieren die einzelnen Typen grafischer Objekte, die als Drawing-Objekt auf dem Server abgelegt sind. Neben einfachen grafischen Objekten wie Linie, Kreis und Rechteck gehören auch komplexere Komponenten wie der Vote, der Elemente einer Benutzungsoberfläche wie Button und Textarea enthält, zu dieser Gruppe (siehe Abbildung 2.7.2).

Die Klasse `GraphicComponent`

`GraphicComponent` implementiert Methoden, die eine Größenveränderung der Komponenten durch die Interaktion mit der Maus ermöglichen. Durch die Überlagerung der Methoden zur Behandlung der Maus-Events definiert `GraphicComponent` Bereiche, in denen durch Setzen eines entsprechenden Modus Größenveränderung zugelassen wird. `GraphicComponent` unterscheidet die Größenveränderung an den vier Ecken der Komponente und der Größenveränderung an den Seiten der Komponente. Die Möglichkeit, die Größe der Komponente durch

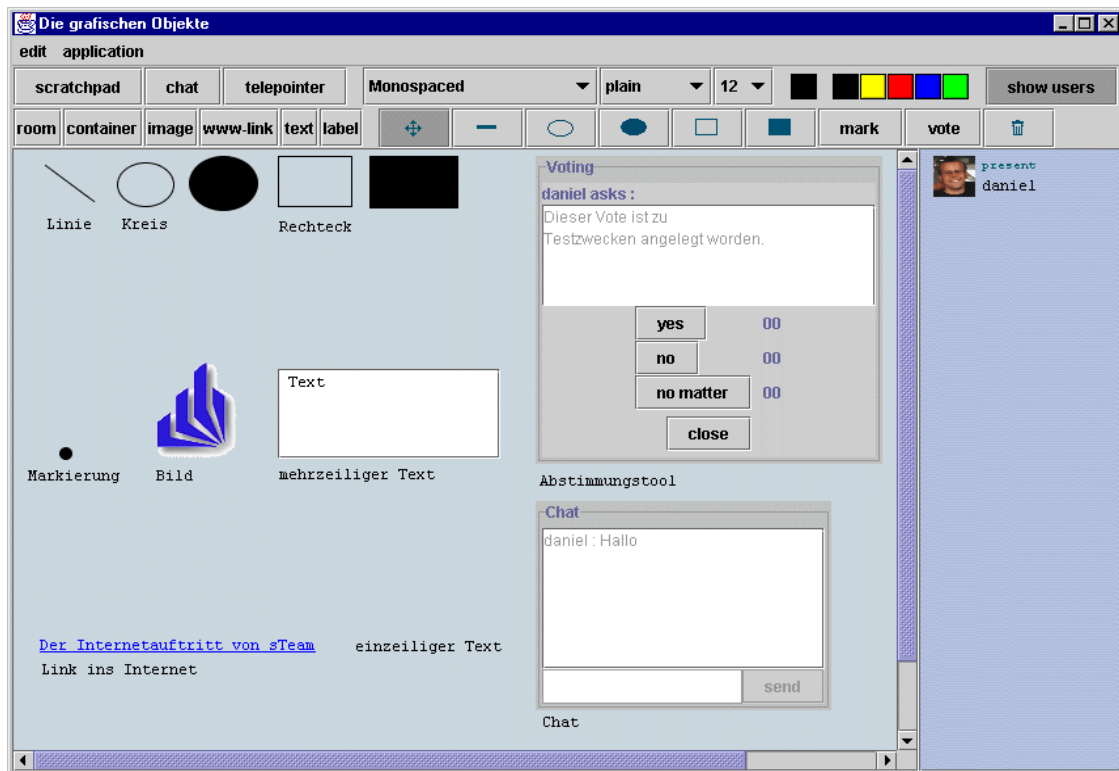


Abbildung 2.4: Darstellung der grafischen Objekte auf der Zeichenfläche

den Benutzer verändern zu lassen kann von den Subklassen nach Bedarf abgeschaltet werden. Um dem Benutzer diese Interaktionsmöglichkeiten deutlich zu machen, wird die Methode `paint()` überschrieben und abhängig von den gesetzten Interaktionsmöglichkeiten werden entsprechende “Knöpfe” gemalt, an denen der Benutzer mit der Maus ziehen kann, um die Größe der Komponente zu verändern.

`GraphicComponent` erlaubt den Subklassen, die Möglichkeiten der Größenveränderungen an ihre Bedürfnisse anzupassen. So lässt sich eine Mindestgröße festlegen, die nicht unterschritten werden kann. Über die Methode `setFlipable()`, die als Parameter einen Boolean-Wert verlangt, wird festgelegt, ob die Größenveränderung fortgesetzt wird, wenn der Benutzer den Komponent in eine negative Größe führen würde. Ist mittels `setFlipable(true)` diese Möglichkeit erlaubt, so ändert der Komponent beim Überschreiten des Nullpunktes oder entsprechend der Maximalhöhe oder -breite seine Ausrichtung und den Modus in der Weise, dass der Benutzer die Größenveränderung weiterhin durchführen kann. Fasst der Benutzer die Komponente an der linken oberen Ecke mit der Maus und zieht sie nach unten, wird die Komponente immer flacher. Zieht der Benutzer die Maus nun weiter nach unten bis über den unteren Rand der Komponente hinweg, ist das weitere Verhalten durch den Status der Variable `flipable` abhängig, die zuvor mittels `setFlipable()` gesetzt wurde. Ist sie auf `false` gesetzt, wird die Komponente so flach, wie es die gesetzte Mindestgröße zulässt. Weiteres ziehen der Maus nach unten hat keine Auswirkungen. In dem Fall, dass `flipable` auf `true` gesetzt wurde, bewirkt ein Ziehen der Maus über den unteren Rand der Komponente, dass diese sich spiegelt und nun durch weiteres Ziehen mit der Maus die linke untere Ecke als Ausgangspunkt für die Größenänderung mit der Maus angesehen wird. Besonders die einfachen grafischen Objekte wie Linie, Kreis und Rechteck profitieren von dieser Funktionalität, da sie so sehr komfortabel verändert werden können.

In `GraphicComponent` wird die Unterstützung der Farbänderung von Komponenten implementiert. Mit der Methode `setColor()` kann die Farbe der Komponenten geändert werden. `GraphicComponent` aktualisiert automatisch das Attribut `DRAWING_COLOR`¹⁴ des Server-Objektes, wenn die Farbe lokal geändert wurde. Durch überschreiben der Methode `updateLocalProperties()` reagiert `GraphicComponent` auf Änderungen des Attributes `DRAWING_COLOR` auf dem Server und ändert die Farbe der lokalen Repräsentation.

Subklassen von `GraphicComponent` können auf zwei Arten erstellt werden. Werden die Komponenten lokal erzeugt, so wird die entsprechende Komponente direkt angelegt und auf der Zeichenfläche positioniert. Sollen Komponenten erstellt werden, um ein Objekt darzustellen, das von einem anderen Benutzer angelegt wurde, so geschieht dies durch den Aufruf der Methode `createGraphicComponent()`. Diese verlangt als Parameter eine Referenz auf das `WhiteBoard`-Objekt auf dem die zu erzeugende Komponente dargestellt werden soll, dem Typ des angelegten Ob-

¹⁴In Anhang ?? befindet sich eine Aufstellung aller Attribute, die von den einzelnen Komponenten des Whiteboards ausgewertet werden.

jekt, eine Instanz von `SteamObject`, die den Zugriff auf das Objekt ermöglicht und eine `HashMap` in der die für die Darstellung der entsprechenden Komponente benötigten Attributwerte gespeichert sind. Anhand des übergebenen Objekttyps erzeugt `GraphicComponent` eine Instanz der entsprechenden Komponente. Mit den in der `HashMap` übergebenen Attributwerte wird der Komponente initialisiert, so stellt er bei der Darstellung auf der Zeichenfläche den aktuellen Zustand des Server-Objektes dar.

Die Klasse `LineComponent`

`LineComponent` repräsentiert eine Linie. Durch Überschreiben der `paint()`-Methode zeichnet `LineComponent` eine Linie in den Ausgabebereich. Die Linie ist durch die Größe der Komponente und den Linienverlauf definiert. Das Attribut `LINE_ATTR_DIRECTION` gibt den Linienverlauf an. Ist `LINE_ATTR_DIRECTION` auf "0" gesetzt, so wird die Linie von der rechten oberen Ecke der Komponenten bis in die linke untere Ecke gezeichnet. Der Bereich, in dem `LineComponent` auf Mausereignisse reagiert, wird durch Überschreiben der Methode `contains()` bis auf wenige Pixel Toleranz dem Verlauf der Linie angepasst.

Die Klassen `CircleComponent` und `FilledCircleComponent`

Eine Instanz von `CircleComponent` stellt einen Kreis dar, eine `FilledCircleComponent`-Instanz repräsentiert einen ausgefüllten Kreis. Der Kreis ist durch die Größe der Komponente definiert, da die Komponente nicht zwingend quadratisch ist, wird ein Oval in den Ausgabebereich gezeichnet, das jeweils Oben, Unten, Rechts und Links die äußere Grenze des Komponenten schneidet.

Die Klassen `RectangleComponent` und `FilledRectangleComponent`

Eine Instanz von `RectangleComponent` stellt ein Rechteck dar, `FilledRectangleComponent` repräsentiert ein ausgefülltes Rechteck. Das Rechteck ist durch die Größe der Komponente definiert. Durch Überschreiben der `paint()`-Methode wird das Rechteck in der entsprechenden Farbe in den Ausgabebereich der Komponente gezeichnet.

Die Klasse `MarkComponent`

Eine Instanz dieser Klasse stellt eine einfache Markierung durch einen Punkt dar. Im Konstruktor wird die Größenveränderbarkeit abgeschaltet und die Größe auf 20x20 Pixel festgelegt. In den Ausgabebereich wird ein gefüllter Kreis gemalt in

der für dieses Objekt gesetzten Farbe gemalt. Wie die bisher beschriebenen grafischen Objekte bietet auch der `MarkComponent` keine weitere Funktionalität an, sondern stellt lediglich die in `GraphicComponent` implementierte Funktionalität zur Verfügung.

Die Klasse `ImageComponent`

`ImageComponent` stellt ein Bild dar. In der `paint()`-Methode gibt die Komponente die Bilddaten, auf die Komponentengröße skaliert, aus. Durch Überschreiben der Methoden zur Verarbeitung der Mausereignisse stellt `ImageComponent` eine proportionale Größenveränderung zur Verfügung. Hält der Benutzer die "Shift"-Taste während einer Größenänderung gedrückt, so wird das Größenverhältnis der Komponente konstant gehalten. Durch entsprechende Kontextmenüeinträge bietet `ImageComponent` den Zugriff auf Methoden, die die Komponentengröße auf die Originalgröße des Bildes zurücksetzen. So lassen sich auch die Bilddaten in dem entsprechenden Format auf der lokalen Festplatte abspeichern.

Wird ein `ImageComponent` lokal erzeugt, so wird die Instanz eines `javax.swing.JFileChooser` angezeigt, über die das entsprechende Bild ausgewählt werden kann. Die Bilddaten werden in einem separaten Objekt auf dem Server gespeichert. Für das Bild-Objekt wird das Attribut `IMAGE_ATTR_IMAGEDATA` gesetzt und so der Zugriff auf die Bilddaten ermöglicht. Da das entsprechende Bild-Objekt bereits in den Raum eingefügt wird, bevor die Bilddaten komplett auf den Server geladen werden können, wird über das Attribut `IMAGE_ATTR_DATA_READY` eine Synchronisierung vorgenommen. Erst nachdem die kompletten Bilddaten auf den Server übertragen wurden wird der Wert dieses Attributes auf "1" geändert. Synchrone Clients, insbesondere der Whiteboard-Client können so mit der Darstellung des Bildes warten, bis die kompletten Daten auf den Server übertragen wurden. Wurden die Bilddaten komplett auf den Server übertragen, so können die Repräsentationen des Bild-Objektes, die das Bild innerhalb anderer Whiteboard-Clients darstellen mit dem Laden der Bilddaten beginnen. Um dem Benutzer eine Rückmeldung über den Ladezustand der Bilddaten zu geben implementiert `ImageComponent` das Interface `SteamConnectionObserver`, das in der `sTeam-API` definiert ist. Die Methode `indicate()` wird nach dem Empfang jedes Datenpaketes aufgerufen. In der `indicate()`-Methode wird ein interner Zähler gesetzt, der den Fortschritt des Ladevorgangs in Prozent angibt. Da die Komponente selbst schon auf der Zeichenfläche dargestellt wird, bevor die Bilddaten komplett auf den Server geladen wurden, wird in dieser Zeit ein ausgefülltes Rechteck als Platzhalter in den Ausgabebereich gezeichnet. Während des Ladevorgangs der Bilddaten wird im unteren Bereich der Komponente ein Fortschrittsbalken, dessen Breite abhängig von dem gesetzten Zähler ist, dargestellt. Sind die Bilddaten komplett geladen, wird das Bild in den Ausgabebereich der Komponente gezeichnet.

Zum Laden der Bilddaten vom Server wird die Methode `getContentOf()` der Klas-

se `GlobalPreferences` aufgerufen. Durch das Laden der Bilddaten über diese Methode erfolgt die Berücksichtigung des Cache beim Laden der Bilddaten. Erst wenn die Bilddaten nicht im Cache zu finden sind, werden sie vom Server geladen.

Die Klasse `TelepointerComponent`

Diese Klasse stellt den Telepointer dar. Jeder Benutzer hat die Möglichkeit, einen Telepointer zu aktivieren. `TelepointerComponent` liegt auf dem obersten Layer. Durch das Überschreiben der Methode `contains()`, in der festgestellt wird ob Mausereignisse im Gültigkeitsbereich eines Komponenten stattfinden, wird erreicht, dass `TelepointerComponent` über alle Mausereignisse unterrichtet wird. Die Position dieses Komponenten wird an die Position des letzten Mausereignisses angepasst. Die Attribute des Telepointer-Objektes auf dem Server werden bei jeder Bewegung der Maus aktualisiert. Dem lokale Benutzer werden die Telepointer aller anderen Benutzer angezeigt, die ihn aktiviert haben. Es gibt keinerlei Zugriffsmöglichkeiten auf die Instanz von `TelepointerComponent`, das anderen Benutzern zugeordnet ist.

`TelepointerComponent` überschreibt die Methoden `processMouseMoved()` und `processMouseDragged()`. Innerhalb dieser Methoden wird die Position der lokalen Komponente an die Mausbewegung angepasst und die Position des Server-Objektes aktualisiert. Anschließend wird das `MouseEvent`-Objekt an einen eventuell vorhandenen Komponenten weitergereicht, der in einem niedrigeren Layer von dem Telepointer überlagert wurde. So können einfache Manipulationen wie das Verschieben, Größenveränderungen und der Zugriff auf das Kontextmenü vorgenommen werden, während der Telepointer aktiviert ist.

Die Klasse `TextComponent`

`TextComponent` stellt die Funktionalität zur Verfügung, die von Komponenten benötigt wird um Text darzustellen. So wird von `TextComponent` der darzustellende Text, sowie die Schriftart, die zur Darstellung des Textes verwendet werden soll verwaltet.

Die Klasse `LabelComponent`

Diese Klasse stellt einen einzeiligen Text dar. Die Größe der Komponente kann nicht durch den Benutzer verändert werden, sondern richtet sich nach dem Platz, der benötigt wird, um den Text in der gesetzten Schriftart darzustellen. `LabelComponent` ist die einfachste Ausprägung einer Komponente, die von `TextComponent` abgeleitet ist.

Die Klasse `TextPlainComponent`

`TextPlainComponent` stellt den Text in einer `javax.swing.JTextArea` dar. Die Größe der Komponente lässt sich vom Benutzer verändern. Durch die Einbettung der `JTextArea` in ein `javax.swing.JScrollPane` kann der Text auch über die Grenzen des Komponenten hinausgehen. In diesem Fall werden Scrollbalken sichtbar, mit deren Hilfe der restliche Text erschlossen werden kann. `TextPlainComponent` fügt einen Eintrag in das Kontextmenü ein, der es ermöglicht, den dargestellten Text auf der lokalen Festplatte zu speichern. Über das Setzen eines entsprechenden Attributes kann der Umbruch des Textes bei Überschreiten der Komponentenbreite erzwungen werden. In diesem Fall passt sich die Breite des Textes an die Komponentenbreite an.

Die Klasse `FreeHandComponent`

Dieser Komponent erlaubt dem Benutzer, eine Frei Hand Zeichnung auf die Zeichenfläche aufzubringen. Die Freihandzeichnung wird als Polygon abgespeichert. Durch diese Vektorisierung ist der `FreeHandComponent` mit einfachen Mitteln größenveränderbar programmiert worden. Zudem wird bei der Behandlung der Mausevents der Verlauf der Linie genau berücksichtigt und somit nur Mausevents von diesem Komponent verarbeitet, die dementsprechend genau auf der dargestellten Linie passieren.

Die Klasse `WWWLinkComponent`

Durch `WWWLinkComponent` wird der Verweis auf eine Internetseite implementiert. In einem speziellen Attribut wird die URL der Internetseite, auf die verwiesen wird, gespeichert. `WWWLinkComponent` stellt einen Dialog zur Verfügung, der beim Erzeugen einer neuen Komponente dieses Typs die URL des Links sowie eine Bezeichnung abfragt. Die Bezeichnung wird als Text in dem Server-Objekt gespeichert. Die Bezeichnung wird in der `paint()`-Methode ausgegeben. Durch das Überschreiben der Methoden `mouseMoved()` und `mousePressed()` wird in der Mitte der Komponente ein Bereich definiert, in dem es durch Klicken möglich ist, sich die gespeicherte URL in einem Browser anzeigen zu lassen. Die Möglichkeit wird dem Benutzer durch eine Veränderung des Mausursors angezeigt.

Die Klasse `AsynchronousComponent`

Die Unabhängigkeit der Komponenten von der Existenz eines Objektes auf dem Server, das durch den Komponent repräsentiert wird, ist in dieser Klasse implementiert. Die von `AsynchronousComponent` abgeleiteten Klassen können als asynchron eingestuft werden, da ihre Darstellung nicht nur von den Attributen des Server-Objektes beeinflusst wird, sondern teilweise unabhängig von dem aktuellen Status

des repräsentierten Objektes ist. Für die Subklasse `ChatComponent`, die den Chat implementiert, existiert z.B. kein Objekt auf der Server-Seite. Dieser Komponent ist nur lokal verfügbar. Da für lokal existierende Instanzen von den Subklassen unter Umständen keine Objekte auf dem Server hinterlegt sind, werden die Subklassen von `AsynchronousComponent` unabhängig von der Layerhierarchie der anderen Komponenten im Vordergrund dargestellt.

Die Klasse `ChatComponent`

Für den `ChatComponent` existiert kein korrespondierendes Objekt in dem dargestellten Raum. Der dargestellte Chat erbt die Funktionalität der Positions- und Größenveränderbarkeit von `GraphicComponent` und wird auf der Zeichenfläche zusammen mit den Komponenten dargestellt. `ChatComponent` aboniert die Chat-Events des aktuell dargestellten Raums und des Benutzers. So wird die Komponente über Nachrichten, die direkt an den lokalen Benutzer gesendet wurden, sowie Nachrichten, die in dem dargestellten Raum abgesetzt wurden, informiert. Zur Darstellung der Nachrichten verwendet `ChatComponent` eine `JTextArea` in der alle Nachrichten untereinander eingefügt werden. Über ein Textfeld lassen sich Nachrichten an die anderen Benutzer schicken. Private Nachrichten können nur an im aktuellen Raum befindliche Benutzer geschickt werden. Sie werden durch Vorstellen von “tell <Benutzername>” im Textfeld an den angegebenen Benutzer geschickt.

Die Klasse `VoteComponent`

Der `VoteComponent` implementiert ein Abstimmungstool. Statt ein grafisches Objekt in der `paint()`-Methode zu zeichnen, werden Bedienelemente zur Verfügung gestellt. `VoteComponent` enthält einen Fragentext, der in einem Textfeld dargestellt wird. Das Attribut `ATTR_VOTE_STATUS` gibt an, welche Interaktionsmöglichkeiten bestehen. Über drei verschiedene Buttons kann mit *Ja*, *Nein* oder *Egal* abgestimmt werden.

Diese Komponente erzeugt erst ein Objekt auf dem Server, wenn der Fragesteller die Frage eingegeben hat und den Vote startet. Die Darstellung des `VoteComponent` erfolgt abhängig davon, ob der lokale Benutzer bereits gewählt hat oder nicht. Hat er bereits an der Abstimmung teilgenommen, werden die zum Wählen benötigten Buttons ausgegraut dargestellt und lassen keine Interaktion mehr zu. Der Vote verbleibt so lange im Raum, bis der Initiator die Abstimmung beendet und das Server-Objekt löscht.

2.7.4 Nicht bekannte Objekte

Während des Wechsels in einen neuen Raum lädt `WhiteBoard` eine Liste, in der alle in diesem Raum enthaltenen Objekte aufgeführt sind. Durch die ständige Weiterentwicklung des Servers und des `Whiteboard-Clients` ist es möglich, dass diese Liste in Zukunft Objekte enthält, für die das `Whiteboard` keine Repräsentation zur Verfügung stellt. In diesem Fall wird eine Instanz der Klasse `unknownComponent` verwendet, um dieses Objekt darzustellen. `unknownComponent` ist von `IconTextComponent` abgeleitet und stellt das unbekannte Objekt als Icon mit dem Namen des Objektes als Unterschrift des icons dar. Durch Überlagern der Methoden `setAttributes()` und `setAttribute()` wird verhindert, dass Attribute des Server-Objektes durch die Interaktion des Benutzers mit der lokalen Repräsentation geändert werden. Operationen, die in `WhiteBoardComponent` über das Kontextmenü verfügbar gemacht werden, sind ausführbar. Die Methode `performCommand()` wurde überlagert, um dem Benutzer vor dem Löschen oder Ausschneiden eines unbekanntes Objektes einen Hinweis zu geben, dass die gerade gewählte Aktion Konsequenzen hat, die mit seiner Version des `Whiteboard-Clients` nicht korrekt dargestellt werden.

Abbildungsverzeichnis

2.1	Die Packagesstruktur	6
2.2	Die Klassenhierarchie	15
2.3	Die Funktionen der einzelnen Klassen	17
2.4	Darstellung der grafischen Objekte auf der Zeichenfläche	26