

Fach: Verteilte Informationsverarbeitung

Studienheft: VII01-03

Projekt: Client-Server-Malprogramm

Dokumenttitel: Phase 2: Entwurf

Dokumentart: Entwurf

Erstellt von: Martin Blankenstein

Erstelldatum: 26.08.2010

Inhaltsverzeichnis:

1 Allgemein.....	2
2 GUI Entwurf	3
2.1 GUI PaintTogether Server	3
2.2 GUI PaintTogether Startanwendung.....	3
2.3 GUI PaintTogether Client	6
3 Architektur	7
3.1 Umwelt.....	7
3.2 Grobe Architektur	8
3.3 Architektur PaintTogetherStartSelector.....	9
3.3.1 der StartClientAdapter	11
3.3.2 der StartServerAdapter.....	12
3.3.3 das StartSelectorPortal	13
3.3.4 die Kernkomponente, der StartSelector	15
3.4 Architektur PaintTogetherServer	17
3.4.1 das ServerPortal	18
3.4.2 der ServerClientAdapter	19
Output am ServerClientAdapter	20
Input am ServerClientAdapter	23
Der ServerClientAdapter als Platine.....	26
3.4.3 die ServerCore	29
3.4.4 Gesamtüberblick aller EBCs im PaintTogetherServer	30
3.5 Architektur PaintTogetherClient.....	31
3.5.1 ClientPortal	32
3.5.2 ClientServerAdapter	34
3.5.3 ClientCore	37
4 Zusammenfassung.....	39

1 Allgemein

Der Entwurf der Client-Server-Anwendung sollte sowohl die GUI (im weiteren Portal genannt) des Clients, die Steuerung des Servers und vor allem die Architektur der gesamten Anwendung beachten und "entwerfen". Probleme die einem eventuell erst bei der Implementierung einfallen würden sollen hier "aufwandsmindernd" schon verhindert werden.

Die Anwendung soll in C# auf Basis des .Net-Framework 3.5 erstellt werden. Als Paradigma für die Entwicklung habe ich mich für die EBC (Event-Based-Components) entschieden. Es handelt sich hierbei um eine spezielle Form der "Komponentenorientierten Softwareentwicklung". Hauptentwickler dieser Idee sind Ralf Westphal und Stefan Lieser. Beide sind selbständige Softwareentwickler und haben ebenfalls die Initiative "Clean-Code-Developer" ins Leben gerufen. Da ich mich selber täglich mit den CCD-Prinzipien und -Praktiken auseinandersetze, werde ich auch bei der Implementierung der Server-Client-Anwendung nach den CCD-Prinzipien und -Praktiken entwickeln. Dazu gehört auch TDD (Test-Driven Development/Design).

Komponentenorientierte Softwareentwicklung

- http://de.wikipedia.org/wiki/Komponentenbasierte_Entwicklung

Event-Based-Components

- <http://ralfw.blogspot.com/2010/02/mustergultig-event-based-components.html>
- <http://ralfw.blogspot.com/2010/02/event-based-components-der-nachste.html>
- <http://ralfw.blogspot.com/2010/02/steckspiele-event-based-components.html>
- <http://ralfw.blogspot.com/2010/02/verbindungsstucke-event-based.html>
- <http://ralfw.blogspot.com/2010/02/bindungsenergie-gedanken-uber-ein.html>
- <http://ralfw.blogspot.com/2010/03/generiert-architekturcompiler-fur-event.html>
- <http://ralfw.blogspot.com/2010/03/tdd-aber-bitte-mit-system.html>
- <http://ralfw.blogspot.com/2010/04/holarchie-event-based-components-fur.html>
- <http://ralfw.blogspot.com/2010/04/geschichtet-event-based-components.html>
- <http://ralfw.blogspot.com/2010/04/feingranular-methoden-als-event-based.html>
- <http://ralfw.blogspot.com/2010/04/kanalisiert-event-based-components-im.html>
- <http://ralfw.blogspot.com/2010/06/remote-communication-mit-event-based.html>

Clean-Code-Developer

- <http://clean-code-developer.de/>

System-Umwelt-Diagramm

- http://www.infforum.de/themen/anforderungsanalyse/thema_RE_systemabgrenzung.htm
- <http://ralfw.blogspot.com/2010/04/holarchie-event-based-components-fur.html>

2 GUI Entwurf

Für die Erstellung der Oberflächen-Entwürfe verwende ich das kostenfreie FireFox mit dem Plugin "Evolus Pencil":

Evolus Pencil

- <http://pencil.evolus.vn>

2.1 GUI PaintTogether Server

Hier handelt es sich um eine Konsole, auf der der Server die Aktivitäten der Clients loggt.

2.2 GUI PaintTogether Startanwendung

In den Anforderungen unter Analyse ist folgendes über die Oberfläche der Startanwendung geschrieben worden:

"Beim Start der Anwendung soll erst eine Auswahl erscheinen, ob man sich bei der malerischen Tätigkeit eines anderen beteiligen möchte, oder ob man eine neue Zeichnung beginnen möchte. Zusätzlich soll beim Start ein Name und eine Farbe ausgewählt werden."

Für das Fenster ergeben sich daraus folgende Anforderungen:

- Name eingeben
- Farbe auswählen
- neue Malerei starten oder sich an anderer Malerei beteiligen

Die folgende Abbildung erfüllt die Anforderungen:

PaintTogether [X]

Willkommen bei PaintTogether. Malen Sie zusammen mit an einer Malerei. Nehmen Sie an einer Malerei teil, oder erstellen Sie eine eigene.

Alias

Farbe

Verhalten:

- bei "Farbe wählen" erscheint der Windows-Farbauswahl-Dialog, mit der dort gewählten Farbe wird der Name im Aliasfeld versehen
 - ☐ es erfolgt keine Prüfung der Farbe (bei Auswahl von weiß sieht man den Alias nicht und später auch nicht die eigenen Striche)
- Solange man keine Farbe ausgewählt hat und/oder keinen Alias angegeben hat, kommt bei Klick auf "An Malerei beteiligen" oder "Neue Malerei beginnen" ein entsprechende Hinweis (Windowsstandardmeldung)

Weiter steht in der Analyse folgendes geschrieben:

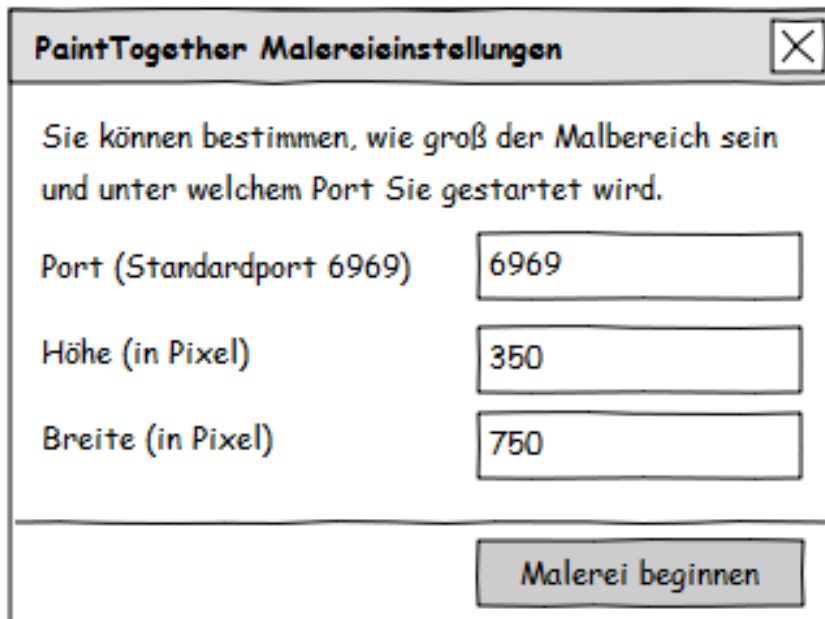
"Wenn man sich entschließt eine neue Malerei zu beginnen, soll man die Größe der Leinwand in Pixeln angeben."

und

"Wie groß darf die Leinwand maximal und wie groß muss sie mindestens sein?"

- *max. 1000*500*
- *min. 500*200"*

Daraus ergibt sich das folgende Fenster, welches sich nach Klick auf "Neue Malerei beginnen" öffnet:



PaintTogether Malereieinstellungen

Sie können bestimmen, wie groß der Malbereich sein und unter welchem Port Sie gestartet wird.

Port (Standardport 6969)

Höhe (in Pixel)

Breite (in Pixel)

Malerei beginnen

Verhalten:

- Bei Klick auf "Malerei beginnen" werden die beiden Eingabefelder gegen die max. und min. Werte aus den Anforderungen validiert und ggf. eine Hinweis angezeigt (Windowsstandardmeldung)
 - ☐ Sind die Einstellungen richtig, startet die Anwendung den PaintTogetherServer und übergibt ihm die Einstellungen (Höhe, Breite, Name). Anschließend wird der Client mit Port und Servername des eben gestarteten Servers sowie den Standardeinstellungen (Name, Farbe) aufgerufen. (Der Anwender hat dann die Konsole für den Server und die Maloberfläche des Clients offen)

Neben dem Starten einer eigenen Malerei gibt es in der Startoption noch die Möglichkeit mit Klick auf "An Malerei beteiligen" sich an einer Malerei zu beteiligen:

"Entschließt man sich zur Beteiligung an einer Malerei, so muss man den Namen oder die IP des Rechners angeben sowie den Port, wo die Malerei gestartet wurde. Ist unter dem angegebenen Ziel eine Malerei aktiv, so erfolgt die Beteiligung an der einen Malerei." und in den Anforderungen wurde ebenfalls geschrieben, dass 'localhost' voreingestellt sein soll.

Für die Angabe des Servers wurde folgendes Formular entworfen:

The screenshot shows a dialog box titled "PaintTogether Malereisuche" with a close button (X) in the top right corner. The main text inside the dialog reads: "Geben Sie hier bitte den Servernamen oder IP des Rechners , auf dem die Malerei begonnen wurde sowie den Port an." Below this text are two input fields. The first field is labeled "Rechner (Name oder IP)" and contains the text "localhost". The second field is labeled "Port (Standardport 6969)" and contains the text "6969". At the bottom of the dialog is a button labeled "Der Malerei beitreten".

Verhalten:

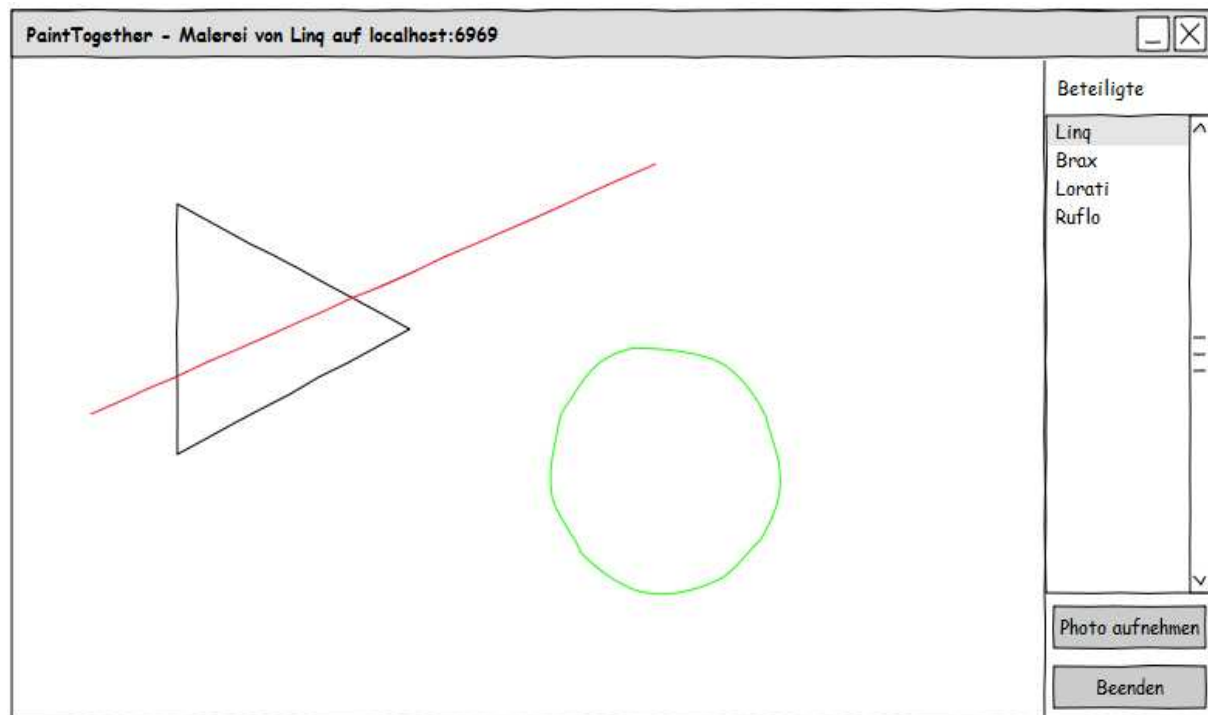
- Klick auf "Der Malerei beitreten"
 - ☐ Prüfung ob der Server da ist, ist er nicht erreichbar kommt ein entsprechender Hinweis
 - ☐ Port prüfen, ist keine Malerei vorhanden kommt ein entsprechender Hinweis
 - ☐ Clientanwendung starten (mit den Parametern: Farbe, Name, Server, Port)

2.3 GUI PaintTogether Client

In dem Fenster der Clientanwendung findet der eigentliche Malvorgang statt. Hier wird also ein Malbereich benötigt. Neben dem Malbereich geht aus den Anforderungen hervor, dass auch alle an der Malerei beteiligten Clients mit ihren Namen und mit ihren Farben dargestellt werden sollen. Hier soll eine einfache Liste mit den Namen der Beteiligten verwendet werden.

Als zusätzliche Aktivitäten bleiben noch das Schließen des Clients und das Festhalten der aktuellen Malerei als extra zu speicherndes Bild. Dazu soll es einen extra Button "Photo aufnehmen" geben.

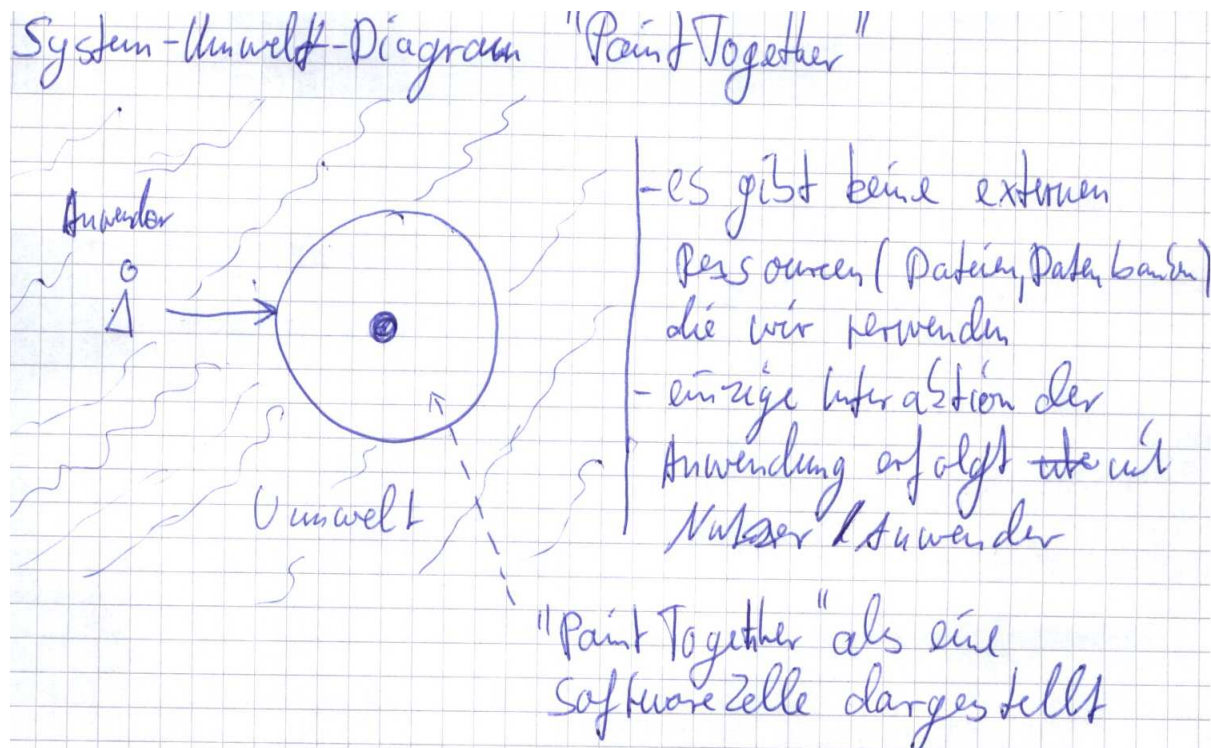
Die entworfene GUI:



3 Architektur

3.1 Umwelt

Zuerst soll anhand eines System-Umwelt-Diagramms herausgefunden werden, von welchen äußeren Ressourcen unsere gesamte Client-Server-Anwendung abhängt (ohne Betriebssystem, .Net-Framework) und wer oder was Sie benutzt.



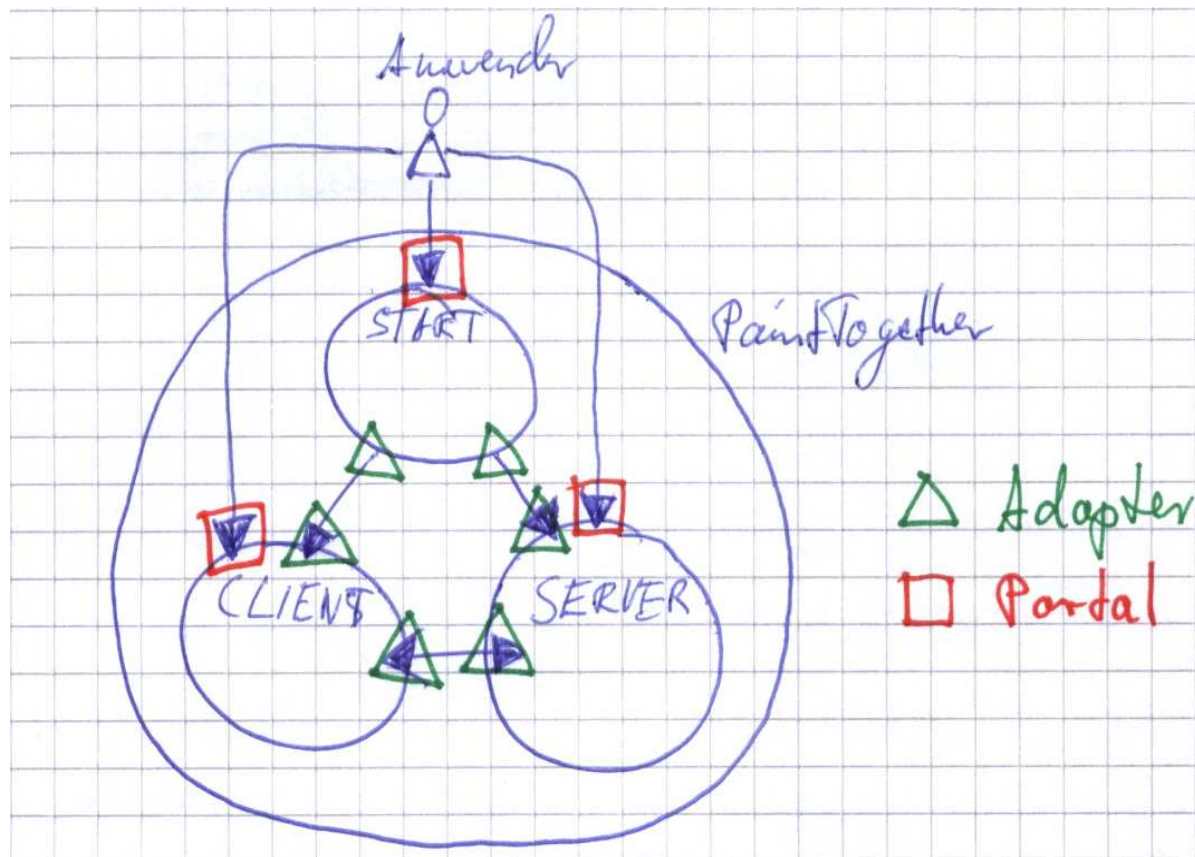
In der Abbildung wird die gesamte Client-Server-Anwendung als eine Softwarezelle dargestellt, mit der der Anwender interagiert. Unsere Anwendung benötigt keinerlei externer Dateien oder Datenbanken für einen korrekten Betrieb (abgesehen vom Betriebssystem).

Die Umwelt-Situation ist damit abgeschlossen.

3.2 Grobe Architektur

Die Software "PaintTogether" besteht wie bereits gesagt nicht nur aus einer einzelnen Anwendung, sondern aus den drei Anwendungen:

- PaintTogetherServer (Server)
- PaintTogetherClient (Client)
- PaintTogether (Startanwendung)



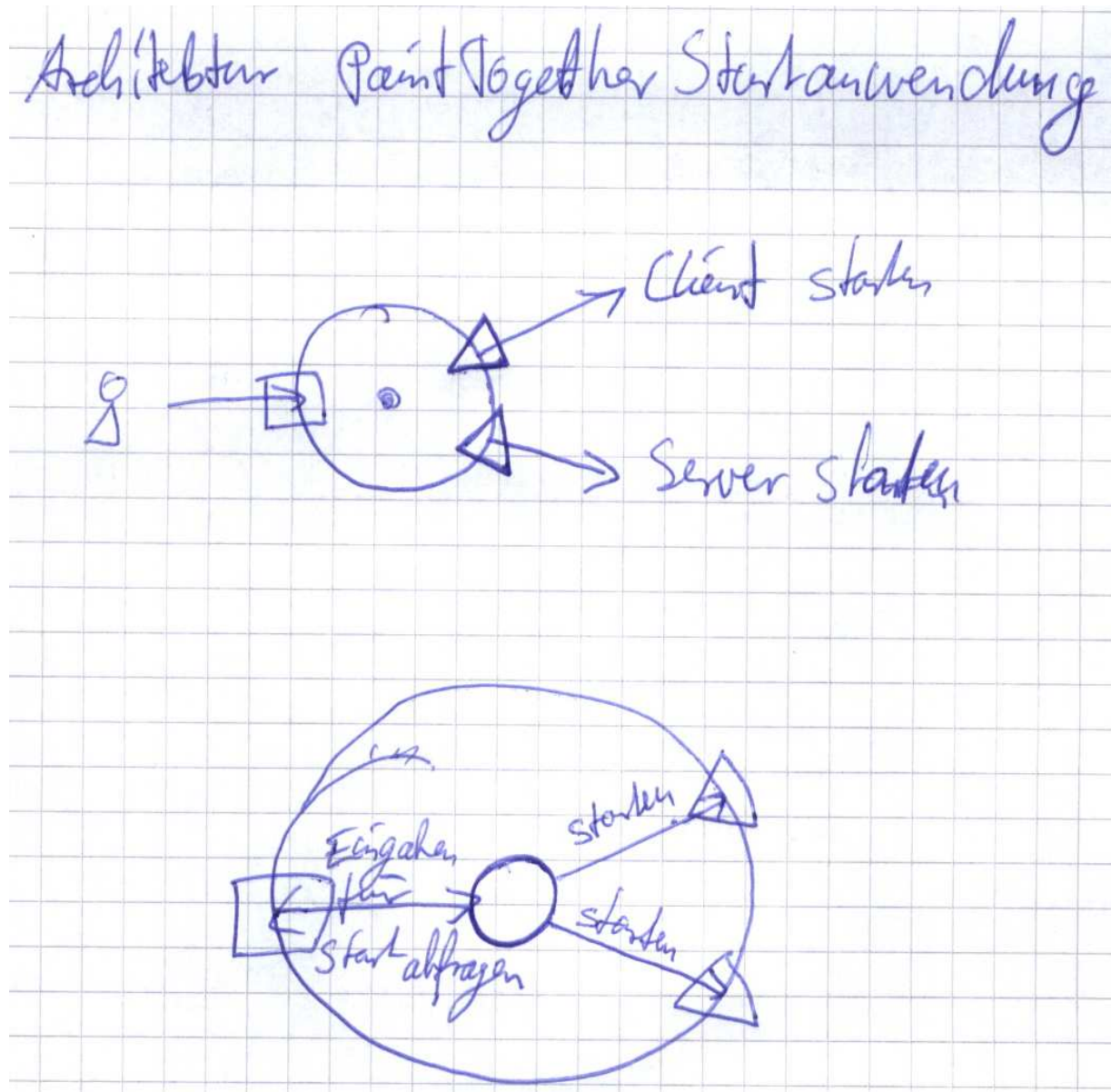
Alle drei Anwendungen sind hier mit ihren Kommunikations- und Interaktionswegen dargestellt. Die hier rot dargestellten Quadrate stellen die Schnittstelle zwischen Anwender und Anwendung dar. Es handelt sich jeweils um ein Portal. Ein Portal kann eine Windows-GUI, eine einfache Konsole, eine Website oder ähnliches sein (einige Portale bestehen aus mehreren Oberflächen). In dem Entwurf der Oberflächen haben wir bereits die GUI der Startanwendung und des Client beschrieben. Bei der ServerGUI wurde eine Konsole angegeben. Es sollen dort lediglich Loginformationen ausgegeben und keine Befehle entgegen genommen werden können. (siehe Analyse - Eigene äußere Anforderungen)

Das für den Client entworfene Fenster ist das Portal für die Clientanwendung. Die beschriebene Konsole des Servers ist das Serverportal und die drei beschriebenen Fenster der Startanwendung bilden zusammen das Portal der Startanwendung.

3.3 Architektur PaintTogetherStartSelector

Die Startanwendung soll dem Nutzer durch die bereits entworfenen Oberflächen die Möglichkeit geben, einfach eine Malerei zu starten bzw. komfortabel einer Malerei beizutreten. Die eigentliche Client-Server-Anwendung funktioniert auch ohne die Startanwendung.

Aus dem Diagramm bei der groben Architektur, wo Portal und Adapter farblich gekennzeichnet sind, wird die Architektur der Startanwendung abgeleitet. Es ergeben sich also ein Portal (besteht aus den drei entworfenen Fenstern), zwei Adapter und einer Kernanwendung:



Wir haben jetzt vier zentrale Komponenten, die wir auf dieser Ebene betrachten. Jede dieser Komponenten soll jetzt als eine EBC (Event-Based-Component) entworfen werden.

Hinweis zu EBCs:

Eine EBC ist eine eigenständige Funktionseinheit. Sie verarbeitet Input und erzeugt Output. Der Input wird über sogenannte Inputpins an die EBC geleitet und der Output verlässt die EBC über

sogenannte Outputpins. Die EBC ist selbstständig und nach außen hin unabhängig. Man kann eine EBC mit einer eigenständigen Nachrichtenverarbeitungszentrale vergleichen, die Nachrichten erhält und eigene Nachrichten verschickt.

Hinweis zu EBCs:

Bei den EBCs ist wesentlich, dass es immer ein Outputpin einer EBC mit einem Inputpin einer anderen EBC verbunden wird. Durch die Verbindung entsteht ein "Draht". Wesentliches Konzept der EBCs ist die Verwendung eines Nachrichtenobjekts für In- und Outputpin. Dies bedeutet, dass die Nachricht zwischen 2 Pins immer aus einem Objekt besteht. Es ist nicht vorgesehen wie bei normalen Methodenaufrufen mehrere Parameter zu übergeben. Ebenso gibt es keine klassischen Rückgabeobjekte. Ein Inputpin ist in der Implementierung immer als Methode ohne Rückgabewert, mit einem Übergabeparameter definiert. Ebenso ist in der Implementierung ein Outputpin immer durch ein Event definiert, welches nur aus dem Nachrichtenobjekt für die Inputmethode besteht. Dabei dürfen die Nachrichtenobjekte keine primitiven Datentypen sein, da diese nicht einfach erweitert werden können. Die Verdrahtung von In- und Outputpin erfolgt auf einer sogenannten Platine, wobei eine Platine selbst eine EBC darstellt. Auf ihr werden die eigenen Pins mit den Pins ihrer internen EBCs verdrahtet. Ein Vorteil der EBCs ist die lose Kopplung, die Erweiterbarkeit, die Austauschbarkeit, die Verständlichkeit sowie die Konzentration auf die Funktionalität der einzelnen Komponenten.

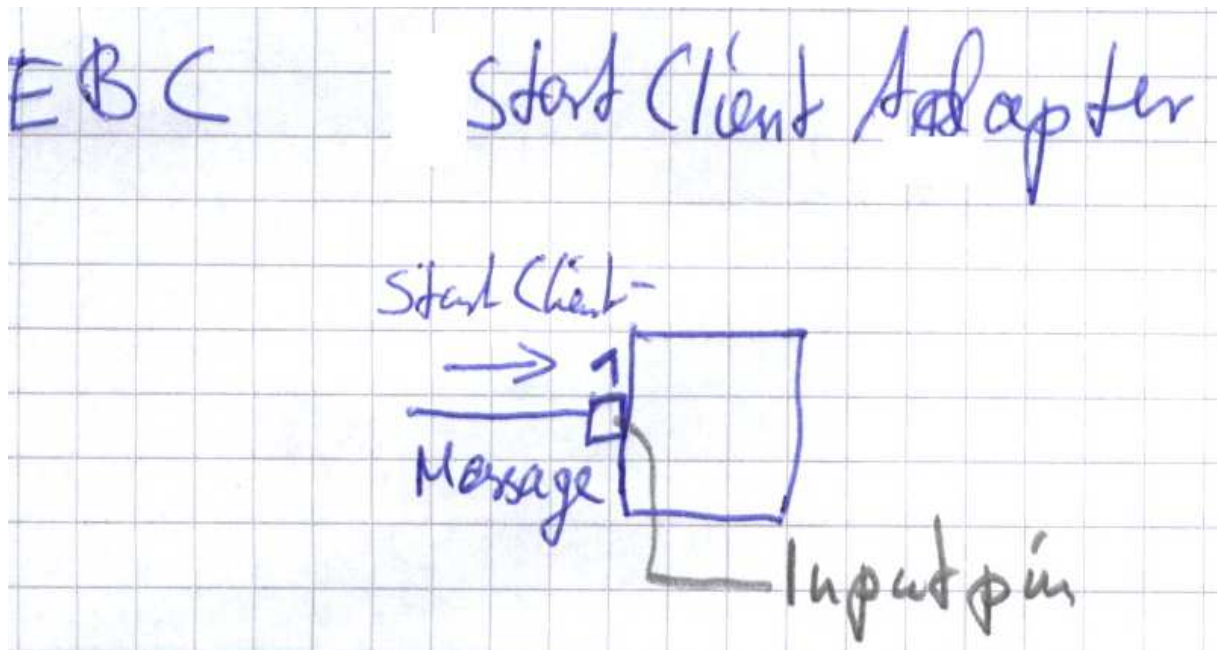
Die einzelnen vier EBCs sollen jetzt erst einmal eigenständig dargestellt/entworfen werden:

3.3.1 der StartClientAdapter

Die Aufgabe dieser EBC besteht lediglich im Start des PaintTogetherClients. Dafür muss ein Inputpin definiert werden, über den das Starten des Clients "beauftragt" werden kann. Das übergebene Nachrichten-Objekt enthält die notwendigen Metadaten für den Start des Client. Wir definieren also auch die in der Abbildung verwendete "StartClientMessage":

StartClientMessage

- | | |
|-----------------------|---|
| + Port : int | Der Port auf dem der Server läuft, mit dem sich der Client verbinden soll |
| + Alias : string | Der Alias mit dem sich der Nutzer auf dem Server anmelden möchte |
| + ServerOrIp : string | Der Rechner, auf dem der PaintTogetherServer läuft |
| + Color : Color | Die Malfarbe, mit der der Nutzer malen möchte |



Hinweis zur Benennung von Nachrichtenklassen bei EBCs:

Eine Nachrichtenklasse endet immer auf Message oder auf Request. Eine Message ist eine einfache in eine Richtung gerichtete Nachricht. Es wird keine direkte Antwort bzw. kein sofortiges Verarbeitungsergebnis erwartet. Die Nachrichtenklassen die auf Request enden erwarten allerdings immer eine Antwort.

3.3.2 der StartServerAdapter

Wie bei der ersten EBC besteht auch die Aufgabe dieser EBC nur im Starten eines Programms. Hier soll ein PaintTogetherServer gestartet werden. Bei der Implementierung könnten durchaus beide Adapter durch eine Implementierung umgesetzt werden. Für die Architektur sind beide aber getrennt zu betrachten.

Hinweis zu EBCs:

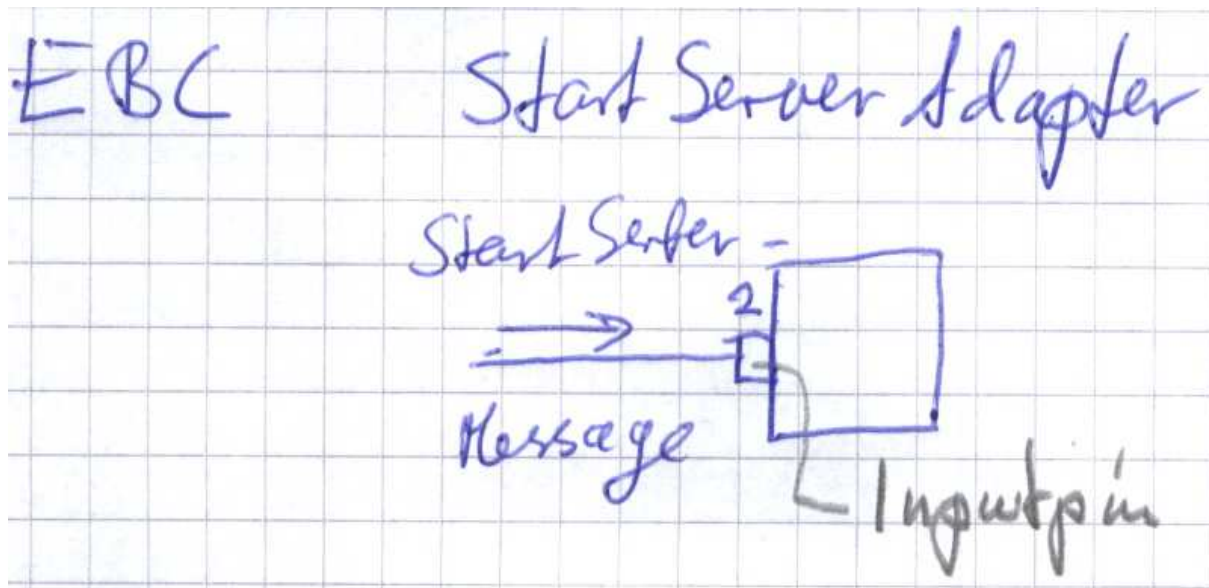
Eine EBC besteht immer aus drei unterschiedlichen Bereichen. Den Nachrichtenklassen, den EBC-Contracts (die Interfaces für die EBCs) und den eigentlichen Implementierungen der EBC-Contracts.

StartServerMessage

+ Port : int
+ Alias : string
angezeigt werden soll
+ Size : Size

Ein freier Port auf dem der Server gestartet werden soll
Der Alias des Nutzers der bei den Clients als Serverbesitzer

Höhe und Breite des Malbereiches der neuen Malerei



3.3.3 das StartSelectorPortal

Das StartSelectorPortal dient der Interaktion mit dem Nutzer. Dieses Portal ist nicht zu verwechseln mit den entworfenen GUIs, aber es besteht im wesentlichen aus den drei entworfenen Oberflächen für die Startanwendung. In der Oberfläche kann der Nutzer folgende Aktionen ausführen:

- Neue Malerei starten
- An Malerei beteiligen

Dies ergibt die zwei wesentlichen Outputpins der StartSelectorPortal-EBC mit den beiden Nachrichtenobjekten "CreateNewPictureMessage" und "ConnectToPictureMessage":

CreateNewPictureMessage

- | | |
|------------------|--|
| + Port : int | Ein freier Port auf dem der Server gestartet werden soll |
| + Alias : string | Der Alias des Nutzers der bei den Clients als Serverbesitzer angezeigt werden soll und mit dem im Client gearbeitet wird |
| + Size : Size | Höhe und Breite des Malbereiches der neuen Malerei |
| + Color : Color | Die Malfarbe, mit der der Nutzer malen möchte |

ConnectToPictureMessage

- | | |
|------------------|---|
| + Port : int | Der Port auf dem der Server läuft, mit dem sich der Client verbinden soll |
| + Alias : string | Der Alias mit dem sich der Nutzer auf dem Server anmelden möchte |

+ ServerOrIp : string Der Rechnername oder die IP des Rechners, auf dem der PaintTogetherServer läuft
+ Color : Color Die Malfarbe, mit der der Nutzer malen möchte

Beide Nachrichten ähneln dabei den Nachrichtendefinitionen der Adaptoren, sind aber nicht identisch mit ihnen. Da in der Nachrichtendefinition für "CreateNewPictureMessage", bei der Eigenschaft "Port" die Bedingung "Ein freier Port" steht, muss der Port vor der Erstellung der Nachricht geprüft worden sein. Die GUI ist für so etwas natürlich nicht zuständig und da wir uns hier im EBC-Entwurf befindet, definiere ich einen zusätzlichen Outputpin, der die Prüfung des Ports beauftragt. Da ich bei dieser Beauftragung ein Ergebnis erwarte, verwende ich kein einfaches Nachrichtenobjekt, sondern einen Nachrichtentyp an dem das Ergebnis gefüllt wird. Um zwischen Nachrichten mit und ohne erwarteter Antwort zu unterscheiden, enden die Namen von Nachrichten mit Antwort mit dem Wort "Request".

TestLoaclPortRequest

+ Port : int Der Port der auf Verfügbarkeit geprüft werden soll
+ Result : bool Ergebnis ob der Port frei ist oder nicht

Auch bei der Nachricht "ConnectToPictureMessage" steht beim "ServerOrIp" "..auf dem der PaintTogetherServer läuft". Läuft heißt, dort ist ein Server erreichbar. Deshalb muss auch hier vorher geprüft worden sein, ob auf dem Server und Port auch wirklich ein PaintToetherServer erreichbar ist. Ich definiere einen weiteren Nachrichtentypen, der ein Ergebnis erwartet:

TestServerRequest

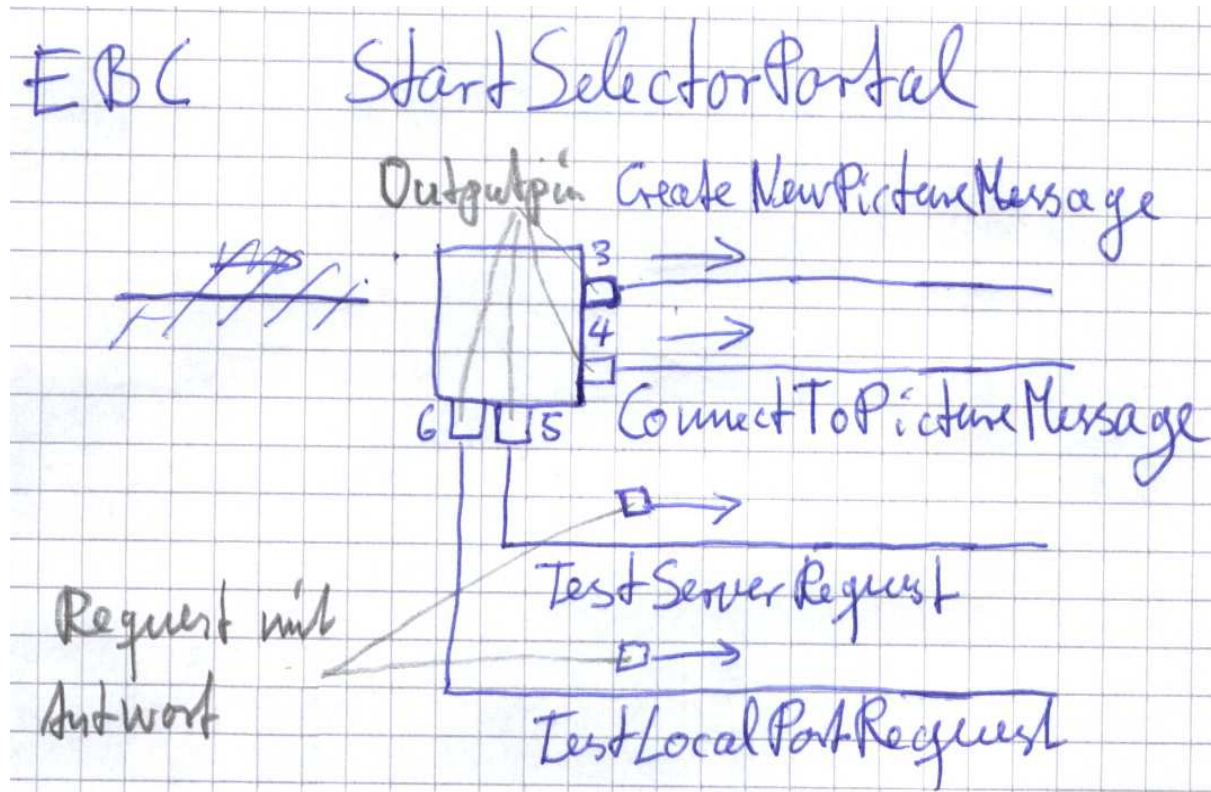
+ Port : int Port, auf dem der PaintTogetherServer eventuell läuft
+ ServerOrIp : string Name oder IP des Rechners auf dem der PaintTogetherServer eventuell läuft
+ Result : bool Ergebnis ob unter dem angegeben Port, des angegeben Rechners eine Verbindung aufgebaut werden kann

Hinweis zu EBCs:

Da zwischen In- und Outputpin nur ein Nachrichtenobjekt verschickt wird, gibt es verschiedene Möglichkeiten eine Antwort zu erhalten. Zwei davon sind:

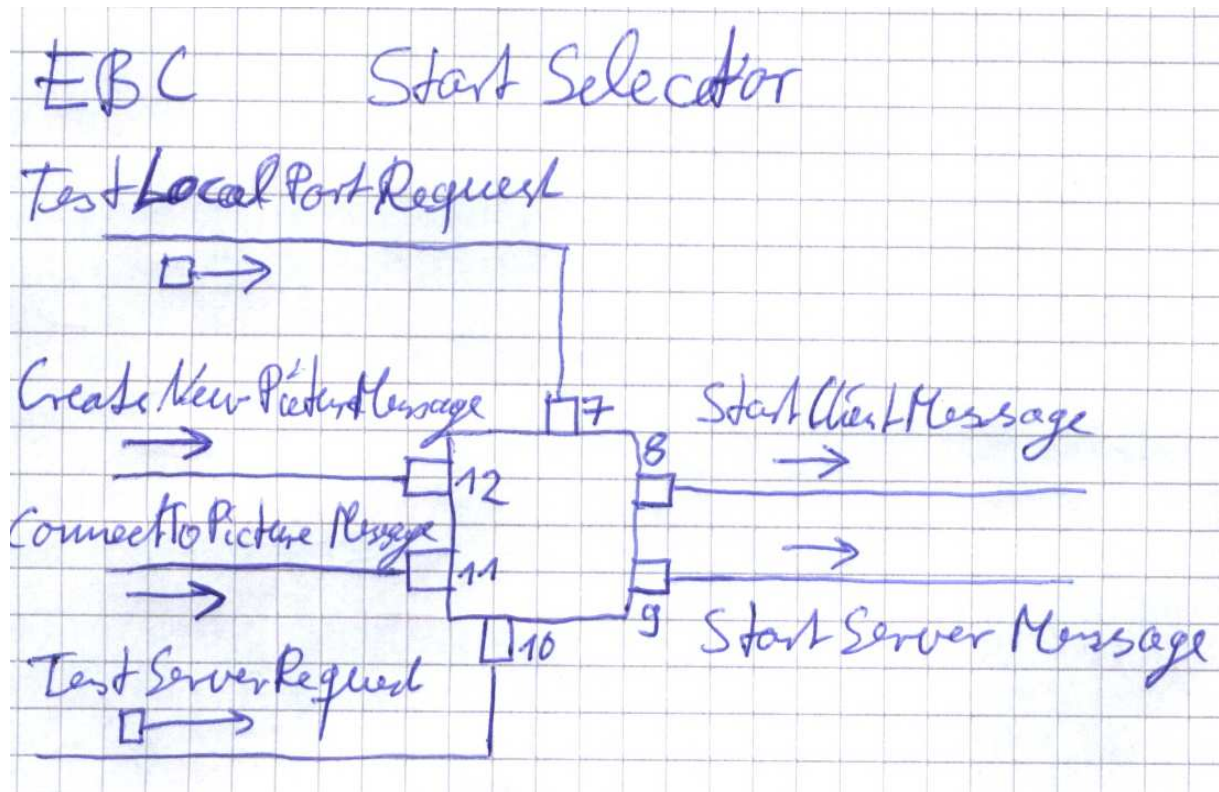
- *zum ersten die von mir verwendete Erweiterung der Nachricht um eine Result-Eigenschaft. Es wird erwartet, dass diese Eigenschaft mit dem richtigen Ergebnis gefüllt wird.*
- *zum zweiten ist das Erweitern der Nachricht um einen Inputpin möglich. Hier wird nicht erwartet, dass die Result-Eigenschaft gefüllt wird, sondern es wird ein Inputpin übergeben, den die EBC, die den Request verarbeitet mit dem Ergebnis aufruft.*

Die aus den beiden beschriebenen Inputpins und den beiden beschriebenen Outputpins entstandene EBC sieht so aus:



3.3.4 die Kernkomponente, der StartSelector

Der StartSelector steht in der am Anfang gezeigten Abbildung in der Mitte der anderen drei schon entworfenen EBCs. Da die StartSelector-EBC mit den drei anderen EBCs verknüpft werden muss, muss sie die passenden Pins besitzen. Die wesentliche Aufgabe ist es somit die Inputpins der drei EBCs durch eigene Outputpins zu "befriedigen" sowie für alle Outputpins der drei EBC einen eigenen, entsprechenden Inputpin bereitzustellen. Die Nachrichtentypen die hier verwendet werden, entsprechen denen der oberen drei EBCs:



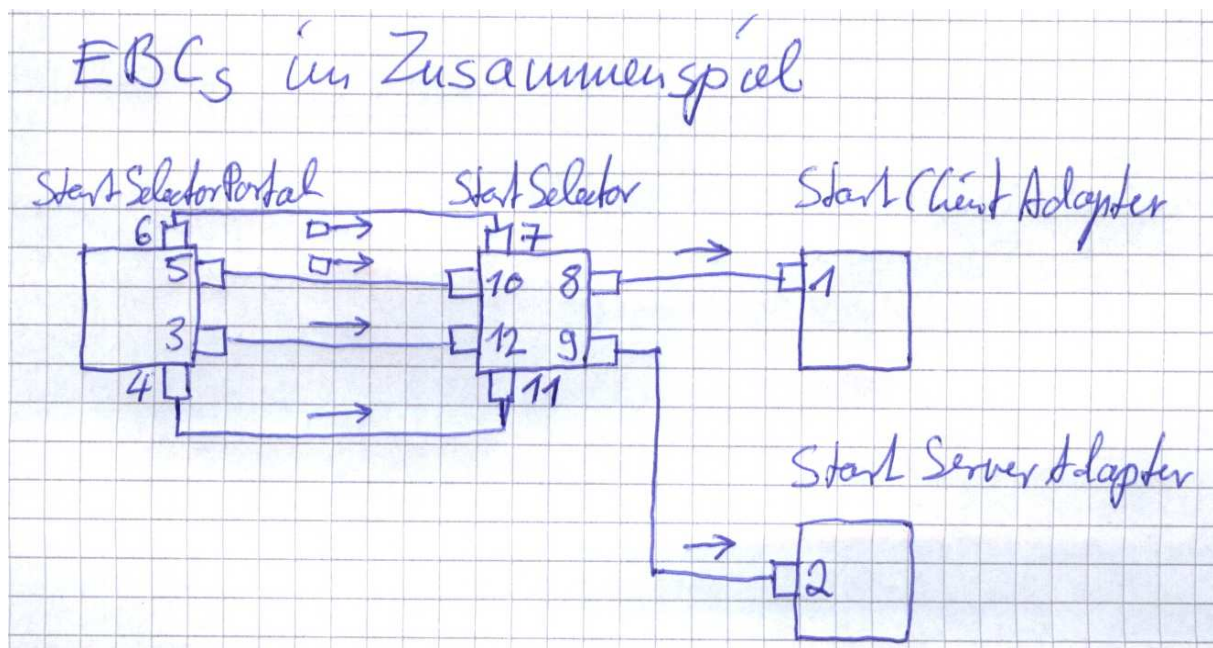
Funktion beim Aufruf der Inputpins:

- 7 : TestLocalPortRequest
 - ☐ Prüft ob der Port frei ist und setzt entsprechend die Antwort
- 10 : TestServerRequest
 - ☐ Prüft ob auf dem angegebenen Rechner, mit dem angegeben Port ein Connect durchgeführt werden kann
 - ☐ Es soll hier nicht extra geprüft werden, ob es sich bei einen PaintTogetherServer handelt, der sich hinter dem erfolgreich verbundenen Port versteckt. Dies wird automatisch geprüft, wenn wir später versuchen einen Client mit dem Server zu verbinden
- 11 : ConnectToPictureMessage
 - ☐ Hier wird die Anfrage direkt an den Outputpin "8 : StartClientMessage" weiter geleitet
- 12 : CreateNewPictureMessage
 - ☐ Hier wird zuerst über den Outputport "9 : StartServerMessage" die Erstellung eines Servers beauftragt, anschließend wird mit "8 : StartClientMessage" die Erstellung des Clients zu dem eben beauftragten Server beauftragt

Zusammenfassung:

- Funktional muss in der Kernanwendung bis auf die Prüfung von Port und externem Rechner nichts gemacht werden, außer der Weiterreichung von Anfragen

Die Architektur der StartAnwendung ist damit abgeschlossen, eine weitere Aufteilung einer der vier hier entworfenen EBCs erachte ich zur Zeit nicht für sinnvoll, da der Funktionumfang minimal ist. Am Ende stelle ich die Startanwendung noch ein Mal mit den verdrahteten EBCs dar. Die Beschriftung der In- und Outputpins erfolgt mit den in den oberen Abbildungen verwendeten Nummern:



3.4 Architektur PaintTogetherServer

Aufgabe des Servers ist die Verwaltung einer Malerei und dessen Clients, außerdem loggt er über ein Portal Informationen für den Anwender.

Wie in der groben Architektur gezeigt, besteht der Server aus einem Portal, einer Kernanwendung und einem Adapter.



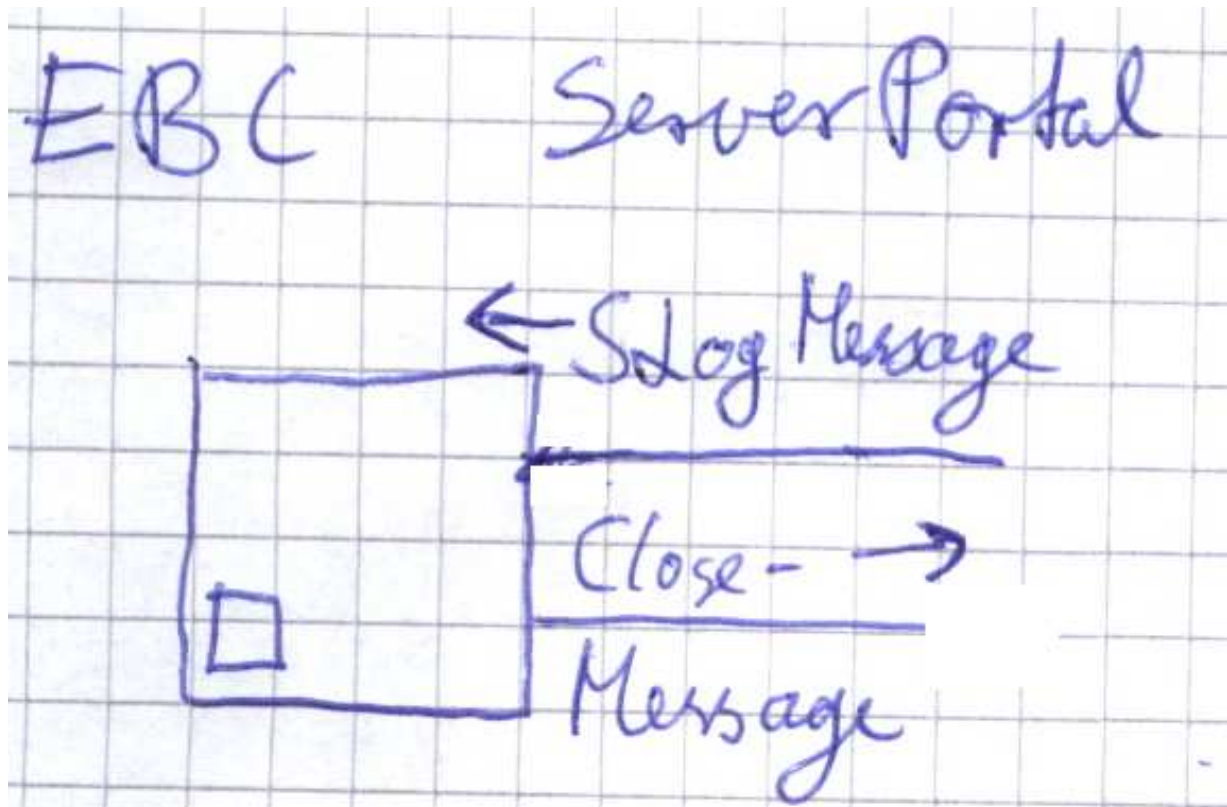
3.4.1 das ServerPortal

Die gesamte Client-Server-Kommunikation liegt hier im Adapter, wie wir gleich sehen werden. Aber vorher ist zu klären, welche Nachrichten werden zwischen dem Portal und der Kernanwendung ausgetauscht und welche Daten zwischen der Kernanwendung und dem Adapter. Dafür verwende ich wieder die EBC-Kasten-Darstellung.

Anmerkung zur Darstellung von EBCs als Rechtecke:

Bei der Darstellung der EBCs in Kasten, kann man unten links in der Ecke noch den Typ der EBC kennzeichnen. Als Typ sind hier Quadrat für ein Portal-EBC, ein Punkt für die Kernanwendung-EBC und ein Dreieck für einen Adapter möglich.

Zuerst entwerfe ich das einfachste der EBCs, das Portal-EBC. Dazu muss klar sein, welchen In- und Output diese EBC verarbeitet. Als Input sind hier nur die Lognachrichten zu sehen, die dem Nutzer angezeigt werden sollen. Als Output gibt es nur das Schließen der Anwendung. Das Schließen der Anwendung sollte hier als Nachricht verschickt werden, damit später alle Client-Verbindungen ordnungsgemäß abgebaut werden können.



Die Nachricht SLogMessage, die hier Input darstellt, enthält nichts weiter als den zu loggenden Text. (Später kann man dies bei Bedarf noch um ein LogLevel erweitern. Das ist aber keine Anforderung)

SLogMessage
+ message : string

Als einziger Output ist hier die CloseMessage vorhanden. Sie hat keinerlei Daten und dient lediglich der Signalisierung das die Anwendung beendet wird.

3.4.2 der ServerClientAdapter

Der Adapter soll die Verbindungen mit den Clients verwalten sowie die Kommunikation mit diesen bewerkstelligen.

Anmerkung zu EBCs:

Bei jeder EBC handelt es sich um eine Funktionseinheit, die eine bestimmte Aufgabe übernimmt. Funktionseinheiten verarbeiten Input und erzeugen Output. Jede Funktionseinheit ist eigenständig und im Normalfall gibt es in der Anwendung bei der Ausführung auch nur eine Instanz jeder Funktionseinheit. Zu der Objektorientierung unterscheidet sich dies insoweit, da

ich dort die Aufgaben von Objekten einer Klasse übernehmen lasse, wobei ich im Normalfall mehrere Objekte einer Klasse habe. Würde ich objektorientiert an die Aufgabe herangehen, dann würde ich mir zuerst über ein Client-Objekt Gedanken machen. Dem Client-Objekt würde ich dann Aufgaben zuordnen, wie das Versenden einer Nachricht und das Empfangen von Nachrichten. Bei dem zu entwerfendem Adapter werden auch die Connections verwaltet, aber nicht in Form von komplexen Objekten. Eine (bisher noch nicht entworfene) Funktionseinheit wird später die Funktionalität zum Empfangen von Nachrichten erhalten und eine andere wird das Senden von Nachrichten übernehmen. Wie genau das dann von statten geht wird sich durch die weitere Konzeption ergeben.

Beim Entwurf der Kommunikation des Adapters muss ich jetzt zum ersten Mal überlegen, was zwischen dem Server und dem Client an Daten ausgetauscht werden muss, um die Anforderungen zu erfüllen. Zur Wahrung der Übersichtlichkeit will ich beim Entwurf der ServerClientAdapter-EBC explizit zwischen eingehenden Nachrichten und ausgehenden Nachrichten unterscheiden.

Output am ServerClientAdapter

Als Output am ServerClientAdapter sind die am ServerClientAdapter über die Clientverbindungen eingegangenen Nachrichten zu interpretieren. Als Nachricht von einem Client ergibt sich als erstes

- Verbindungsaufbau eines Clients mit dem Server

Beim Aufbau der Verbindung mit einem Client benötigen wir die Informationen Alias sowie die Malfarbe des Anwenders der Clientanwendung. Diese beiden Informationen beinhaltet somit auch die erste Outputnachricht.

Als weitere eingehende Nachricht gibt es noch das

- Malen

eines verbundenen Clients. Hier muss definiert werden, was eine "habe gemalt-Information" vom Client beinhaltet. Da ich ein festes Raster habe, muss es sich um Koordinaten handeln. Hier besteht nun noch die Möglichkeit nicht jeden einzelnen bemalten Pixel der Malerei als eigene Nachricht zu sehen, sondern alle in einer bestimmten Zeit gemalten Pixel als Nachricht zu verschicken. Für diese Client-Server-Anwendung verzichte ich auf das Zusammenfassen von mehreren bemalten Pixeln und definiere, dass jeder bemalte Pixel als Nachricht vom Client an den Server gesendet werden muss. Als Inhalt der "gemalt"-Nachricht ist somit die X-Y Position zu sehen. Beim Entwerfen der AdapterEBC nehme ich aber auch die Farbe des bemalten Pixels mit in den AdapterOutput.

Nun bleibt noch das Trennen einer Client-Server-Verbindung, was ich als

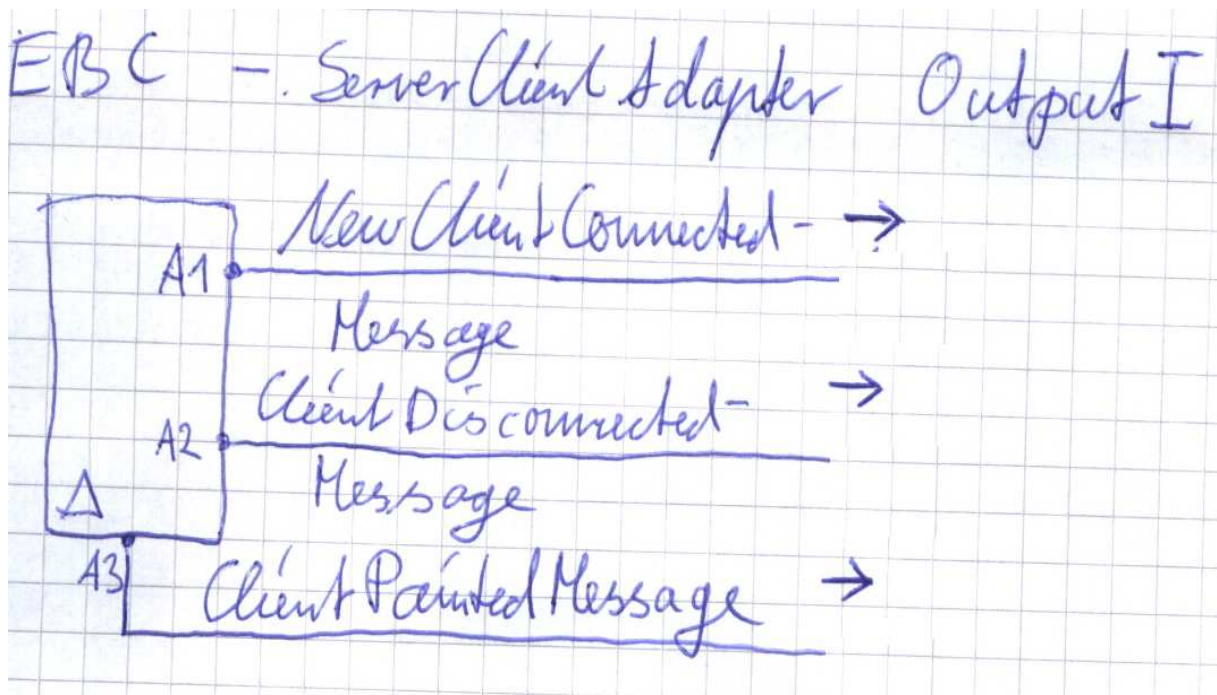
- Verlassen des Servers

bezeichne. Als Output des Adapters ist somit die Information über das Verschwinden eines Clients zu sehen. Hier sind die gleichen Information wie beim Verbindungsaufbau mit zu übergeben (Alias und Farbe).

Notiz:

Wenn sich einmal mehrere Clients mit dem selben Alias und der selben Farbe verbinden, dann weiß die Kernanwendung nicht mehr, welcher der Clients sich jetzt eigentlich beendet hat. Dies ist jedoch unproblematisch für die Kernanwendung. Diese benötigt nur die Information über X Client mit Alias+Farbe und entfernt einfach einen der identischen Clients. In den Anforderungen ist nicht definiert, das später in der GUI zwischen "Torsten-rot" und "Torsten-rot" unterschieden werden muss. Deshalb müssen wir auch außerhalb des Adapters erst einmal keine Unterscheidung anhand einer ID oder ähnlichem treffen.

Die AdapterEBC mit den ersten ausgehenden Nachrichten im Überblick:



NewClientConnectedMessage

+ alias : String

+ color : Color

ClientDisconnectedMessage

+ alias : String

+ color : Color

ClientPaintedMessage

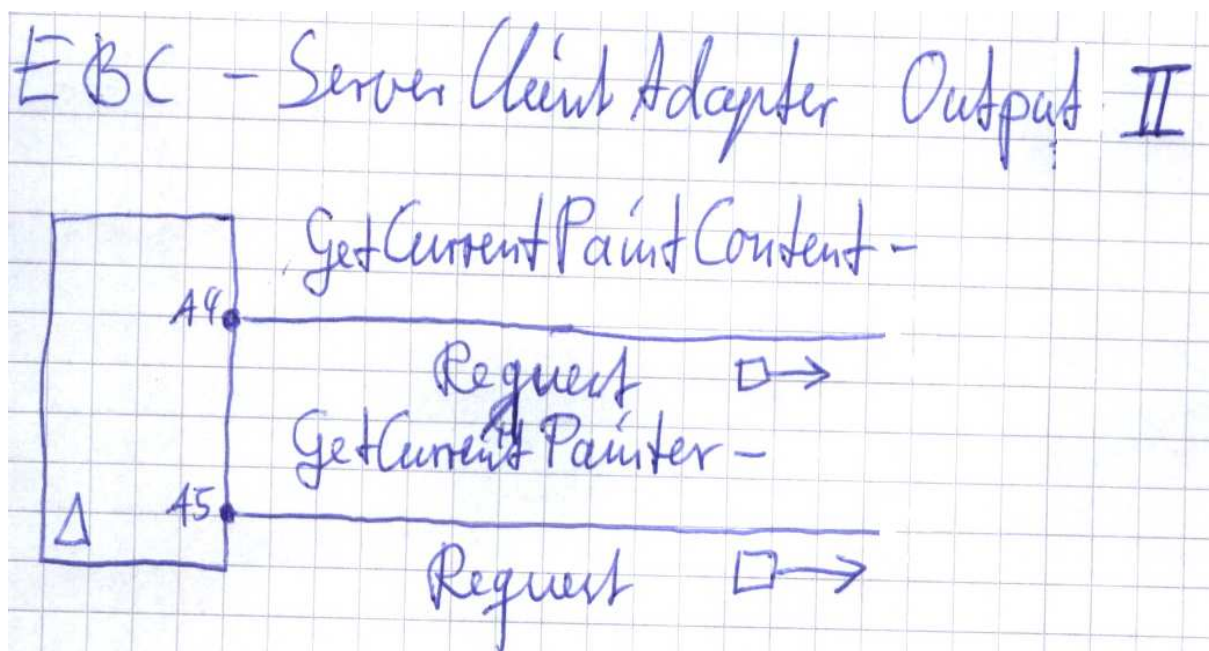
+ color : Color

+ point : Point

Da bei dem Aufbau der Clientverbindung auch Daten an den Client gesendet werden müssen (Bisheriger Stand der Malerei (inkl. Spielfeldgröße) + Alias und Farbe der schon beteiligten Clients), muss es noch weitere Outputnachrichten geben. Da der Adapter den aktuellen Malinhalt und die Beteiligtenliste nicht kennt, müssen diese Informationen ermittelt werden. Dafür erstelle ich zwei weitere Outputs in Form von Requests, die eine Antwort erwarten. Erforderliche Nachrichten:

- Spielfeldinhalt + Größe ermitteln
- Beteiligte Personen ermitteln

Die beiden Nachrichten in EBC-Darstellung:



GetCurrentPaintConetentRequest

+ result : Bitmap

als Bitmap, Größe ist hier Eigenschaft der Bitmap

Aktueller Malbereich

GetCurrentPainterRequest

+ result : KeyValuePair<string, Color>[] Beteiligtenliste in Form eines Arrays aus KeyValuePair

Input am ServerClientAdapter

Als Input des Adapters sind alle Anfragen aus der Kernanwendung zu sehen. Aus den Anforderungen geht hervor, dass alle Beteiligten die von anderen Personen gemalten Striche/Punkte sehen. Dazu muss also eine

- Information über das Malen (Koordinaten + Farbe) an alle Beteiligten erfolgen.

Notiz:

Der Adapter sendet nicht von alleine an alle Clients das "Gemalt", was er ja selber von einem Client gesendet bekommt. Dafür ist der Adapter nicht zuständig. Es ist wichtig die Zuständigkeiten und Funktionen zu unterscheiden. So liegt die Entscheidung die Clients über den Malvorgang eines Clients zu informieren bei der Kernanwendung. Man kann zum Beispiel später in der Kernanwendung einbauen, das Bemalen von Punkten die schon bemalt wurden zu ignorieren. Der Adapter hat als einzige Aufgaben die Clients zu verwalten und Nachrichten entsprechend weiterzuleiten.

Ein weiterer Input ist die Aufforderung

- alle Clientverbindungen zu beenden

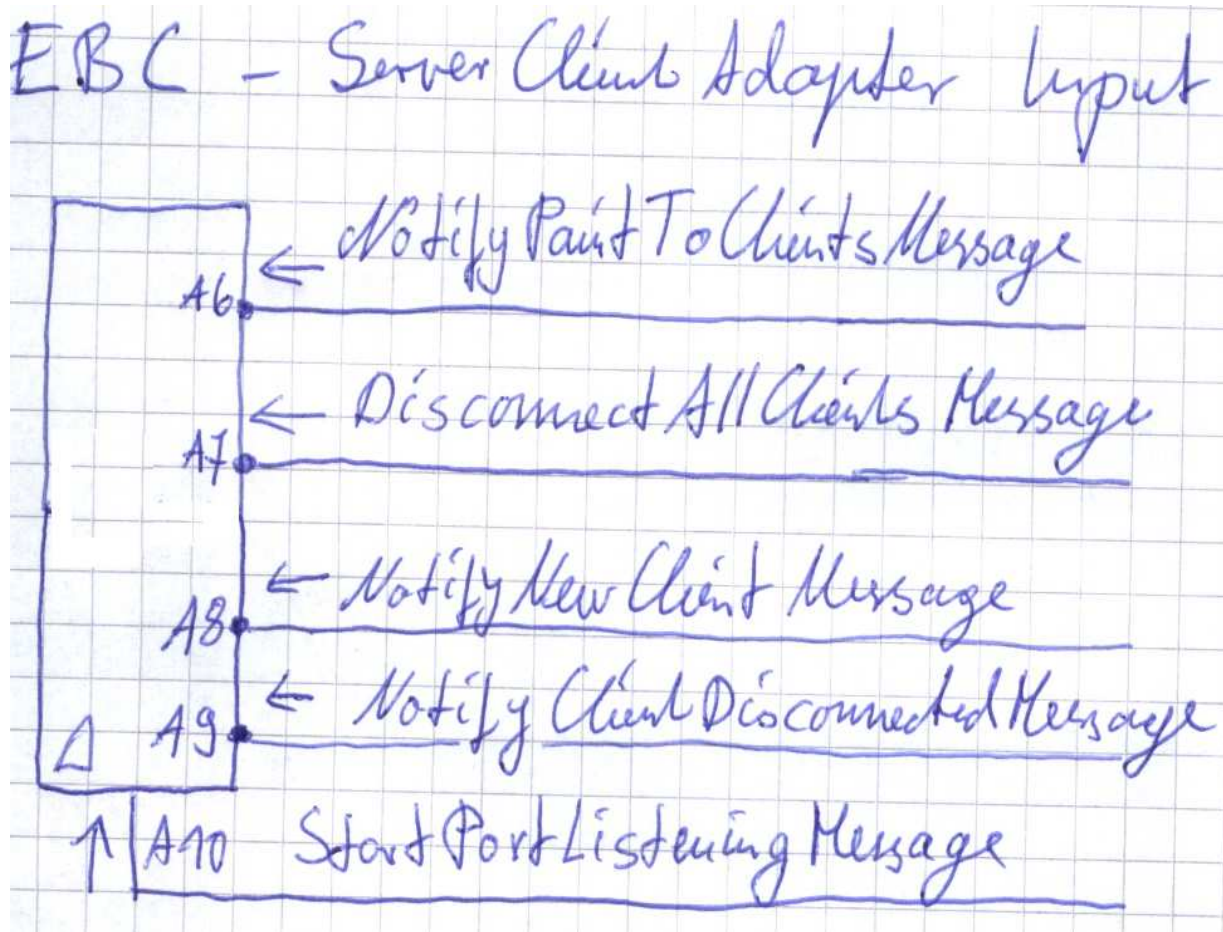
Dies ist beim Beenden der Anwendung erforderlich.

Zusätzlich müssen alle Clients noch über neue Clients und Beendete Clients informiert werden, damit die Beteiligtenliste der Clients aktuell bleibt. Alternativ könnte der Client sich auch alle X Sekunden die aktuelle Beteiligtenliste neu laden. Ich ziehe die Informierung über neue Beteiligte vor, was zwei weitere Inputs für den Adapter erzeugt:

- Clients über einen neuen Beteiligten informieren
- Clients über das Verlassen eines Beteiligten informieren

Zuletzt fehlt noch der initiale Input. Für das Aufbauen von eingehenden Clientverbindungen muss der Server zuerst einen Port öffnen. Diese Aufgabe lege ich auch in den Adapter:

- StartPortListening



NotifyPaintToClientsMessage

+ color : Color

+ point : Point

DisconnectAllClientsMessage

NotifyNewClientMessage

+ alias : String

+ color : Color

NotifyClientDisconnectedMessage

+ alias : String

+ color : Color

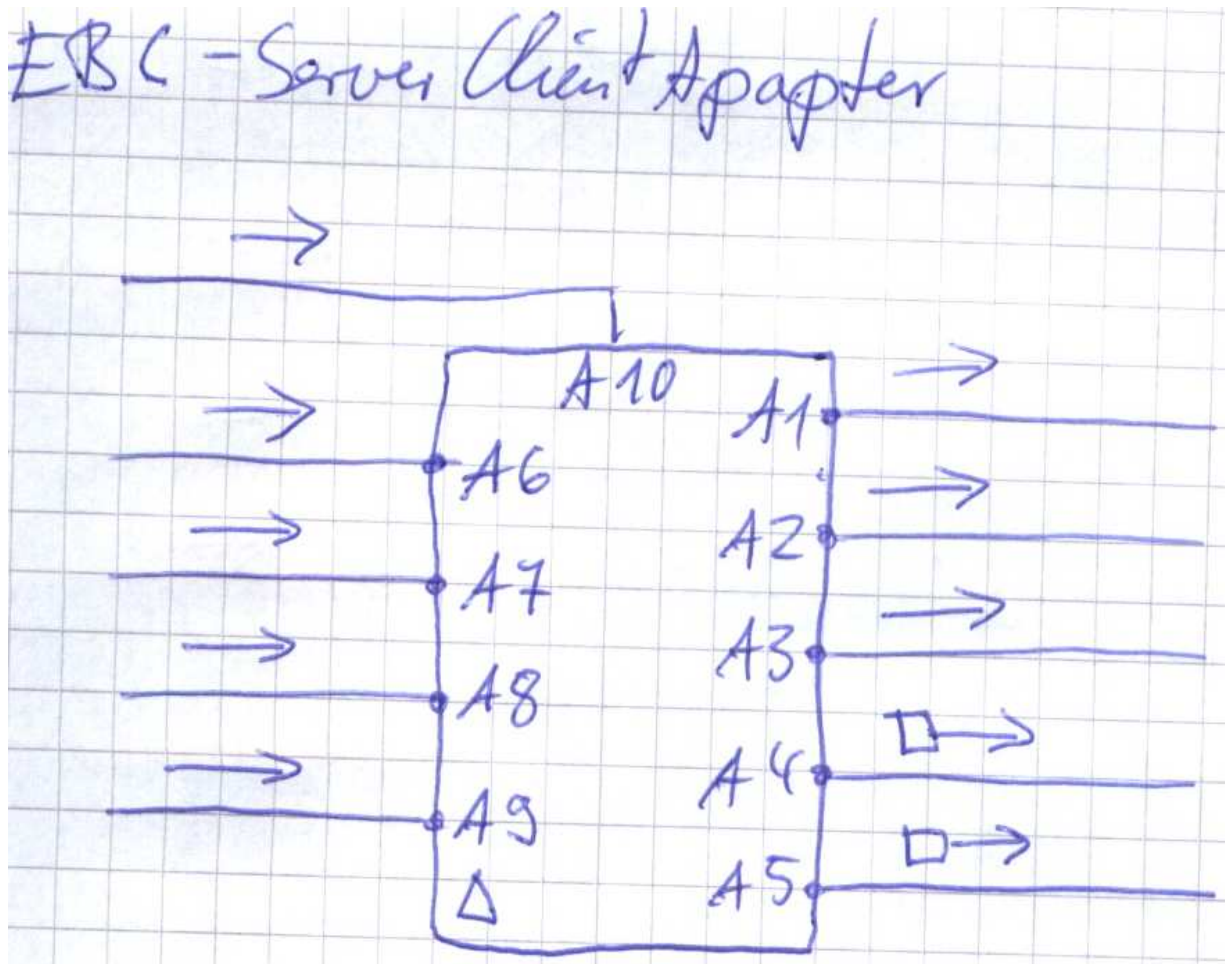
StartPortListeningMessage

+ port : int

Der zu überwachende Port

Die Disconnect-Message besitzt keinerlei Daten. Es wird nur zur Schließung der Verbindungen aufgerufen.

Der Entwurf der Adapter-EBC sieht jetzt so aus (Die oben entworfenen In- und Outputs wurden von A1-A10 durchnummeriert - A=Adapter)



Die Clientverwaltung erscheint mir zusammen mit der Nachrichtenerstellung und -versendung sowie dem Empfangen von Nachrichten zu viel für einen einzelnen Baustein. Aus diesem Grund bohre ich den Adapter auf und mache ihn zu einer Platine, auf der ich verschiedenen Bausteinen die einzelnen Aufgaben übergebe, die sich im Ganzen in der Adapter-EBC verbergen.

Anmerkung zu EBCs:

Ein EBC kann eine Platine oder ein Baustein sein. Dabei übernehmen nur die Bausteine im engeren Sinne die Realisierung der Funktionen. Die Platinen haben selber keinerlei Funktionalität. Sie setzen sich aus mehreren EBCs (Bausteine oder Platinen) zusammen, um eine größere Funktionseinheit darzustellen. Der Adapter wird eine solche Platine zu der ich im

folgenden versuche die einzelnen Bausteine heraus zu kristallisieren. Nach außen hin sieht man dem Adapter bei der späteren Verwendung nicht an, ob es sich um einen Baustein oder eine Platine handelt. Im Endeffekt ist eine Platine auch nur ein Baustein, der wieder auf eine Platine gesteckt werden kann.

Der ServerClientAdapter als Platine

Prinzipiell wissen wir jetzt welche Nachrichten und Daten zwischen unserem Server und Client versendet werden. Auf welche Art und Weise die Nachrichten versendet werden, außer einfach über TCP, ist noch offen. Für die genauere Betrachtung des Adapter ist es aber wesentlich. Bei der Kommunikation über TCP werden erst einmal nur Bytes übertragen. Wie diese Bytes zu interpretieren sind, kommt auf den Sender an. Da es für PaintTogether keine Vorgaben bei der Kommunikation gibt, wähle ich in Bytes umgewandelte XML-Zeichenketten als Nachrichtinhalt. Im Adapter gibt es also einen Nachrichtenkontext, zum Beispiel "Client hat Punkt bemalt". Dieses Objekt soll jetzt an alle Clients geschickt werden. Da ich aber nur Bytes schicken kann, muss vorher eine Umwandlung erfolgen, die eindeutig umkehrbar ist. Dafür verwende ich XML-Serialisierung und -Deserialisierung. Also das Nachrichtenobjekt serialisieren wir vor dem Senden in eine Zeichenkette um, die nichts als ein XML enthält und kodieren dieses in Bytes. Beim Empfangen werden wir alle erhaltenen Bytes einfach wieder in eine Zeichenkette dekodieren und diese Zeichenkette dann wieder in ein Nachrichtenobjekt deserialisieren.

Bestimmung des Socket-Verbindungstyps:

Bei der Verwendung von Sockets muss man neben dem Verbindungstypen TCP auch noch die Art der Datenübertragung definieren. Da ich bei der Malanwendung eine dauerhafte Verbindung zwischen Server und Client benötige verwende ich den Übertragungstypen "Stream". Mit ihm ist die gleichzeitige, beidseitige und verlustfreie Datenübertragung zwischen zwei Sockets möglich.

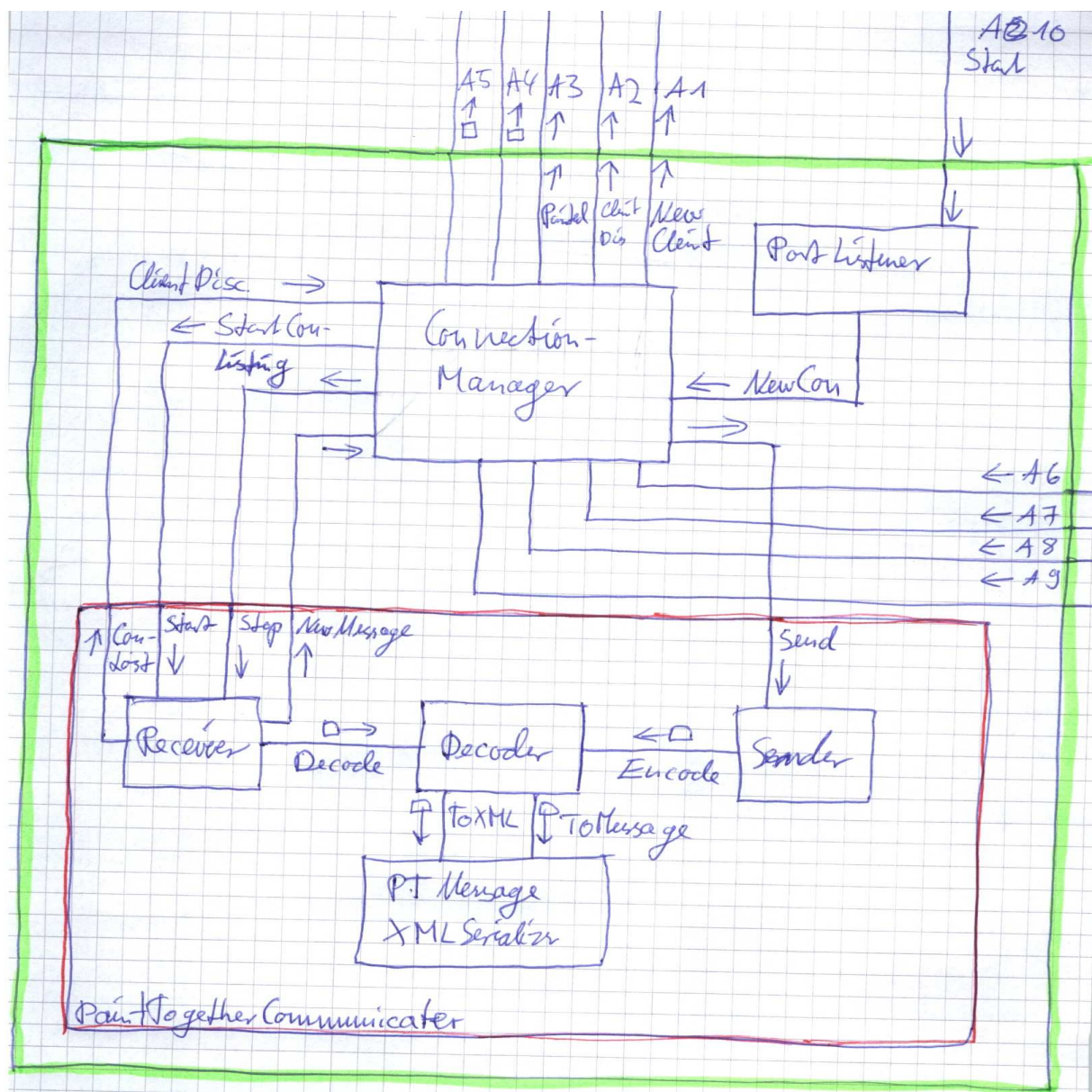
Für die Aufteilung der EBCs in verschiedene kleinere Funktionseinheiten, müssen die verschiedenen Funktionen extrahieren. Zu den Aufgaben die der ServerClientAdapter übernimmt zählen:

- Verarbeiten von neuen Clientverbindungen
- Kodieren einer Nachricht für das Versenden über eine Socket-Verbindung
- Dekodieren einer Nachricht von einer Socket-Verbindung
- Auf Nachrichten einer Clientverbindung warten
- Clientverbindung beenden
- Nachricht an Client senden

Die folgende Platine ist entstanden:

- grün umrandet ist der ServerClientAdapter

- ☐ hier sieht man auch die schon vorher definierten In- und Outputs (A1-A10)
- rot umrandet ist der beim Entwurf entstandene PaintTogetherCommunicater, der hier als eine EBC betrachtet wird und die folgenden Aufgaben übernimmt
 - ☐ Senden eines Nachrichtenobjekts an einen Client
 - ☐ Überwachung einer Clientverbindung starten und stoppen
 - ☐ Empfangene Daten von überwachten Clientverbindungen in Nachrichten umwandeln und über empfangene Nachrichten informieren
 - ☐ Über den Verbindungsverlust einer Clientverbindung informieren



Die ServerClientAdapter-Platine besteht jetzt aus dem

- PortListener
 - ☐ Aufgabe ist Öffnen und Überwachung des Serverport auf eingehende Clientverbindungen
- ConnectionManager
 - ☐ Verwaltet die Clientverbindungen
 - ☐ Beauftrag das Versenden und Überwachen von Clientverbindungen

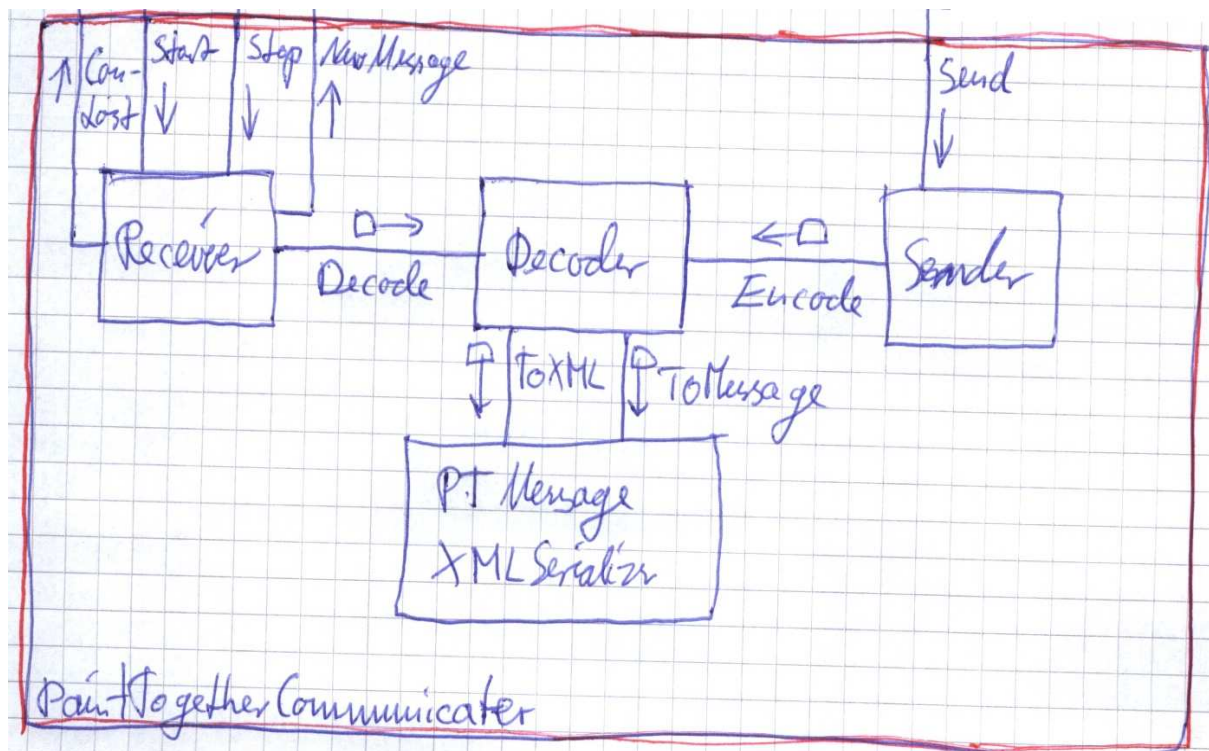
und dem schon beschriebenen

- PaintTogetherCommunicater.

Aus diesem ist beim Entwerfen ebenfalls eine Platine entstanden und besteht aus den vier EBCs

- Receiver
- Sender
- Decoder
- PTMessageXMLSerializer

Nun haben wir mit dem PaintTogetherCommunicater eine EBC, deren Funktionalitäten wir auch im PaintTogetherClient brauchen. Diese EBC wird später ein eigenes Projekt, was vom Client und Server verwendet werden kann.



Bei den Nachrichten wird hier für das Senden und Starten/Stoppen einer Überwachung immer die Connection mitgegeben, für die die Aufgabe übernommen werden soll. Das gilt auch für das

Signalisieren über den Empfang einer Nachricht, wo ohne das Mitsenden der Connection keine Zuordnung möglich wäre. Die einzelnen Nachrichten die innerhalb des ServerClientAdapterEBC und die innerhalb des PaintTogetherCommunicater ausgetauscht werden, werden hier nicht detailliert aufgeschrieben, da diese für den Entwurf der EBC nicht von Bedeutung sind.

3.4.3 die ServerCore

Zwischen dem ServerPortal und dem ServerClientAdapter befindet sich die ServerCore. Hier ist die Hauptfunktion des PaintTogetherServers angesiedelt, das

- Verwalten des Malbereichs
- Verwalten der Alias+Farbe für Beteiligte

Der Entwurf der Nachrichten, die mit dem ServerPortal und mit dem ServerClientAdapter ausgetauscht werden, wurde schon gemacht. Denn es muss lediglich für alle Pins des ServerPortal sowie für alle Pins des ServerClientAdapter ein GegenPIN entworfen werden. Dies ist nötig, um später die drei EBCs miteinander "verdrahten" zu können. Als einzig neuer Pin ist ein StartInputPin zu sehen, der dafür sorgt, dass der Malbereich erzeugt und der ServerClientAdapter mit dem Port gestartet wird.

Zudem ist bisher noch die Art des Loggings offen geblieben. Da die ServerCoreEBC dem Portal Log-Nachrichten schicken muss, definiere ich jetzt das verwendete Loggingsystem. Ich verwende für das Loggen im Server "log4net" von Apache. Somit kann in der gesamten Anwendung über log4net geloggt werden. Damit die Lognachrichten dann auch von der ServerCoreEBC gesendet werden, muss ich die Lognachrichten mit einem eigenen Logappender abfangen und auf dem Outputpin für Lognachrichten an das Portal senden.

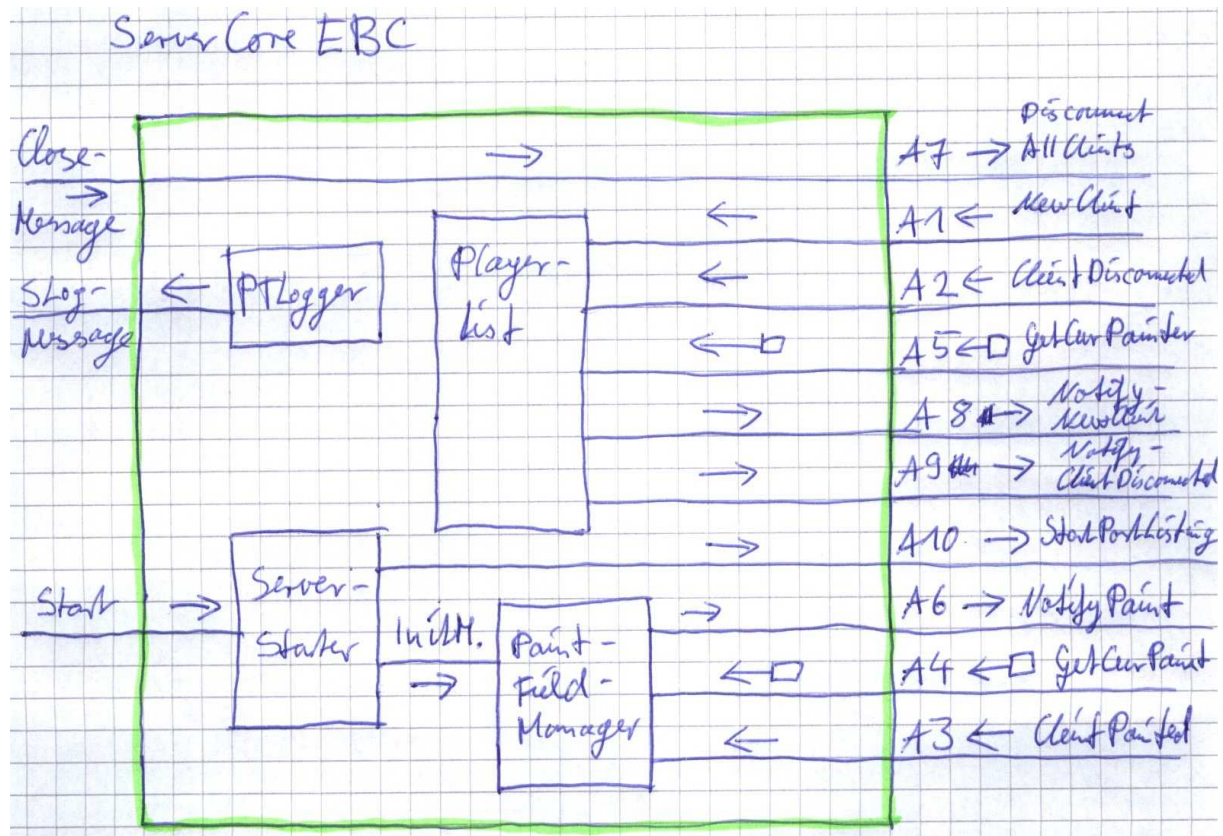
log4net

- <http://logging.apache.org/log4net/index.html>

Jetzt entwerfe ich eine Platine für die ServerCoreEBC die jetzt die Aufgaben

- Verwaltung des Malbereichs
- Verwalten der Alias+Farbe für Beteiligte
- Initialisieren des Malbereichs und Auffordern zum Starten des ServerClientAdapter mit einem Port
- Abfangen aller log4net-Nachrichten und Senden einer Lognachricht auf dem entsprechenden Outputpin

Es ergibt sich die folgende Platine:

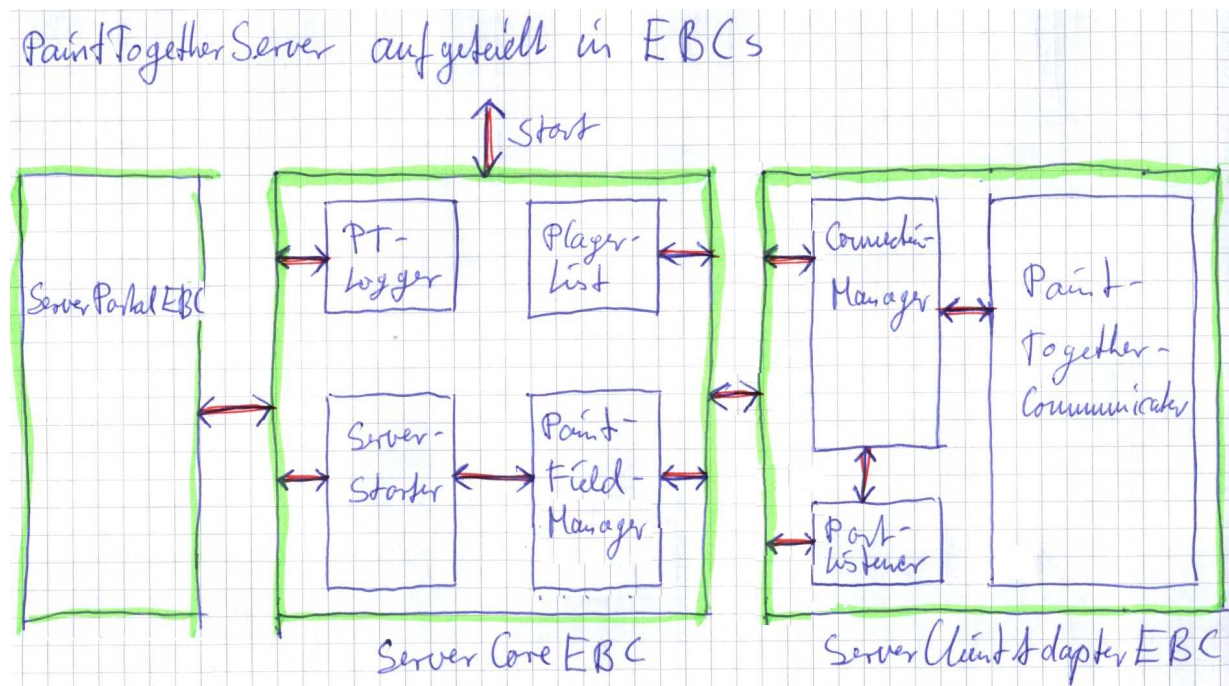


Die äußeren Nachrichten wurden alle schon beim Entwurf des ServerPortals und des ServerClientAdapters beschrieben, bis auf die Startnachricht. Diese besteht aus dem Port, dem Alias und der Spielfeldgröße. Die Beschreibung der inneren Nachrichten halte ich für den Entwurf nicht wesentlich, da es sich nicht um eine Feinkonzeption handelt.

Anmerkung:

Als erstes muss später der Inputpin "Start" des ServerCoreEBCs bedient werden, da erst der Malbereich aufgebaut und der ServerClientAdapter gestartet werden muss. Ich hätte auch statt eines Inputpins einfach im Konstruktor des ServerStarterEBCs die Parameter erwarten können. Beide Varianten halte ich für richtig, habe mich aber für die Signalisierung über einen Pin entschieden.

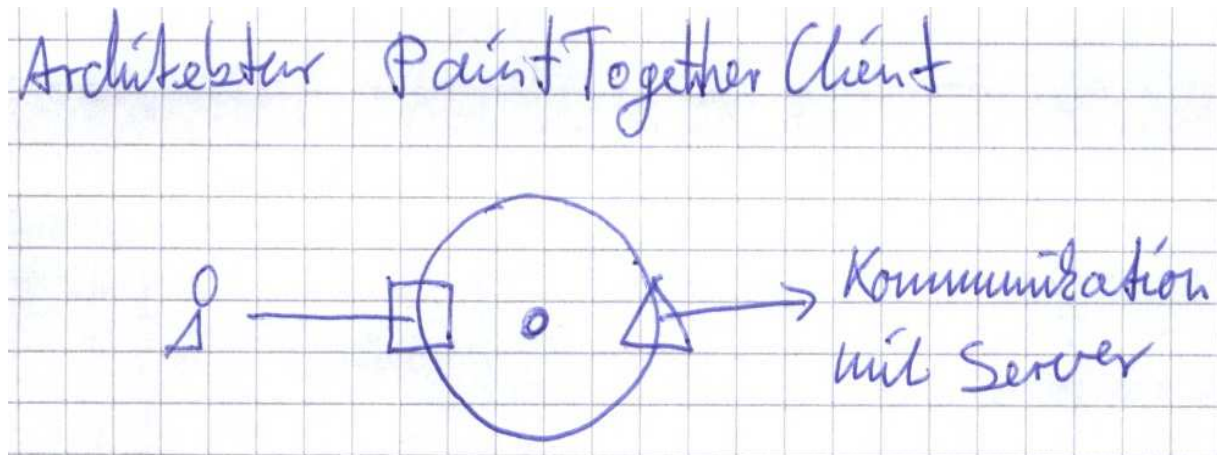
3.4.4 Gesamtüberblick aller EBCs im PaintTogetherServer



Grün umrandet sind die drei Haupt-EBCs. Rot ist die vereinfachte Kommunikation zwischen den EBCs dargestellt. Dabei ist hier immer ein Hin- und Rückpfeil gemalt worden, obwohl nicht immer in beide Richtungen Nachrichten ausgetauscht werden. Diese Abbildung soll lediglich die entworfenen EBCs in Kooperation darstellen und so zu einem besseren Überblick der entworfenen EBCs verhelfen. Die PaintTogetherCommunicaterEBC habe ich nicht unterteilt dargestellt, da ich diese in einem eigenen Projekt bearbeiten möchte und sie hier einfach vom ServerClientAdapterEBC verwendet wird.

3.5 Architektur PaintTogetherClient

Der PaintTogetherClient stellt für den Anwender die "fassbare" Software da. Hier wird der eigentliche Anwendungsnutzen geschaffen, denn in der Oberfläche des Clients (siehe GUI Entwürfe) findet der Malprozess statt. Zu Beginn des Entwurfs fangen wir wieder mit der groben Struktur der Anwendung an. Hier unterscheiden wir wieder zwischen den Portalen, den Adaptern und der eigentlich verarbeitenden Software.



Wie der PaintTogetherServer besteht auch der Client aus einem Adapter, einem Portal und einer Kernanwendung. Für den Adapter können wir schon auf das Wissen über den Server zugreifen und den dortigen PaintTogetherCommunicater verwenden. Bei dem Portal müssen wir alle Funktionen der bereits entworfenen Oberfläche bedienen.

3.5.1 ClientPortal

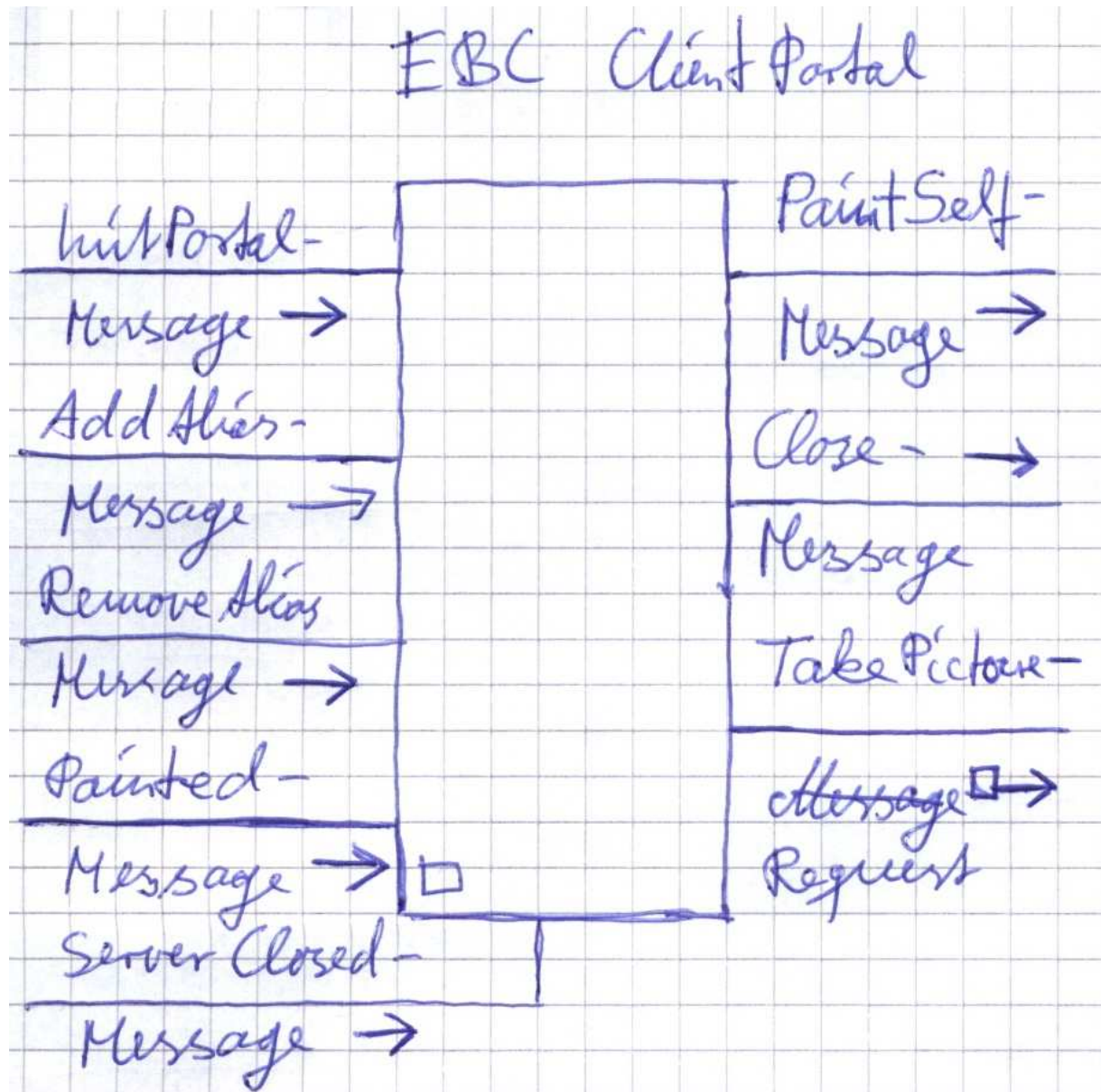
Das ClientPortal besteht aus nur einer Oberfläche. Um es als EBC zu entwerfen sind die Aufgaben herauszufinden und als In- und Outputs zu definieren. Zuerst die Aufgaben die die Oberfläche erfüllen muss, ohne das der Anwender etwas macht:

- Anzeige des Alias, Rechners und Ports des Servers
- Anzeige der Alias und Farben der Beteiligten
- Anzeige des aktuellen Malbereichs sowie der eigenen Änderungen und Änderungen durch die Beteiligten
- Aktualisierung der Aliasliste

Nun die Aufgaben, die durch den Anwender in der Oberfläche vorgenommen werden:

- Malen
- Schließen der Anwendung
- Photo aufnehmen

Die gesammelten Aufgaben werden jetzt durch In- und Outputpins am ClientPortal abgedeckt:

**Hinweis:**

Die einzelnen Nachrichteninhalte schreibe ich hier nicht noch einmal auf, da sie für den Entwurf nicht wesentlich sind. Zu Beginn des Entwurfes hielt ich es hingegen für sinnvoll um ein Gefühl für die Arbeit mit Event-Based-Components zu entwickeln.

Wir haben jetzt fünf einfache Inputs und drei Outputs. Selbsterklärend sind die Nachrichten NICHT, weshalb ich die wichtigsten beschreiben muss:

- InitPortalMessage -> Enthält den Malbereich als BitMap, den Namen/Port/Alias des Servers, die Größe des Malbereichs
- Initialisiert die GUI

- Add- und RemoveAliasMessage bestehen aus Alias und Farbe
 - ☐ Bei InitPortalMessage habe ich absichtlich nicht die Beteiligten angegeben, da diese einfach durch das mehrfache Bedienen von AddAlias übergeben werden können, das spart einen Inputpin
- PaintSelfMessage habe ich "PaintSelfMessage" genannt, um diese Nachricht zwischen der Nachricht "PaintedMessage" unterscheiden zu können, inhaltlich bestehen beide aus Farbe und einer Position
- TakePictureMessage besteht aus dem Zieldateinamen und erwartet als Result einen Text zur Anzeige für den Nutzer (Die Daten über den Malbereich müssen hier nicht mitgeschickt werden)

3.5.2 ClientServerAdapter

Der ClientServerAdapter ist für die Kommunikation mit dem Server vorgesehen und soll alle zu sendenden Nachrichten an den Server weitergeben bzw. alle anfallenden von diesem Empfangen. Die Kernanwendung soll somit nichts von der Art der Datenübermittlung wissen. Wie die Kommunikation funktioniert und welche Daten wir auf welche Weise übertragen wurde bereits beim Entwurf des PaintTogetherServer beschrieben. Die Komplexität des ClientServerAdapter ist zudem niedriger als die Komplexität des ServerClientAdapters im PaintTogetherServer, da wir nur eine Socket-Verbindung verwalten müssen. Damit wir die EBC entwerfen können sind auch hier wieder zuerst die Aufgaben zu benennen:

- Verbindungsaufbau zu einem angegebenen Server+Port
- Überwachen der Verbindung um Nachrichten zu Empfangen
- Senden von Nachrichten über die Verbindung
- Informieren über empfangene Nachrichten
- Umwandeln von Nachrichten nach XML und umgedreht
- Kodieren von XML in Bytes und Dekodieren der Bytes in XML
- Informieren des Anwenders über Verbindungsverlust zum Server

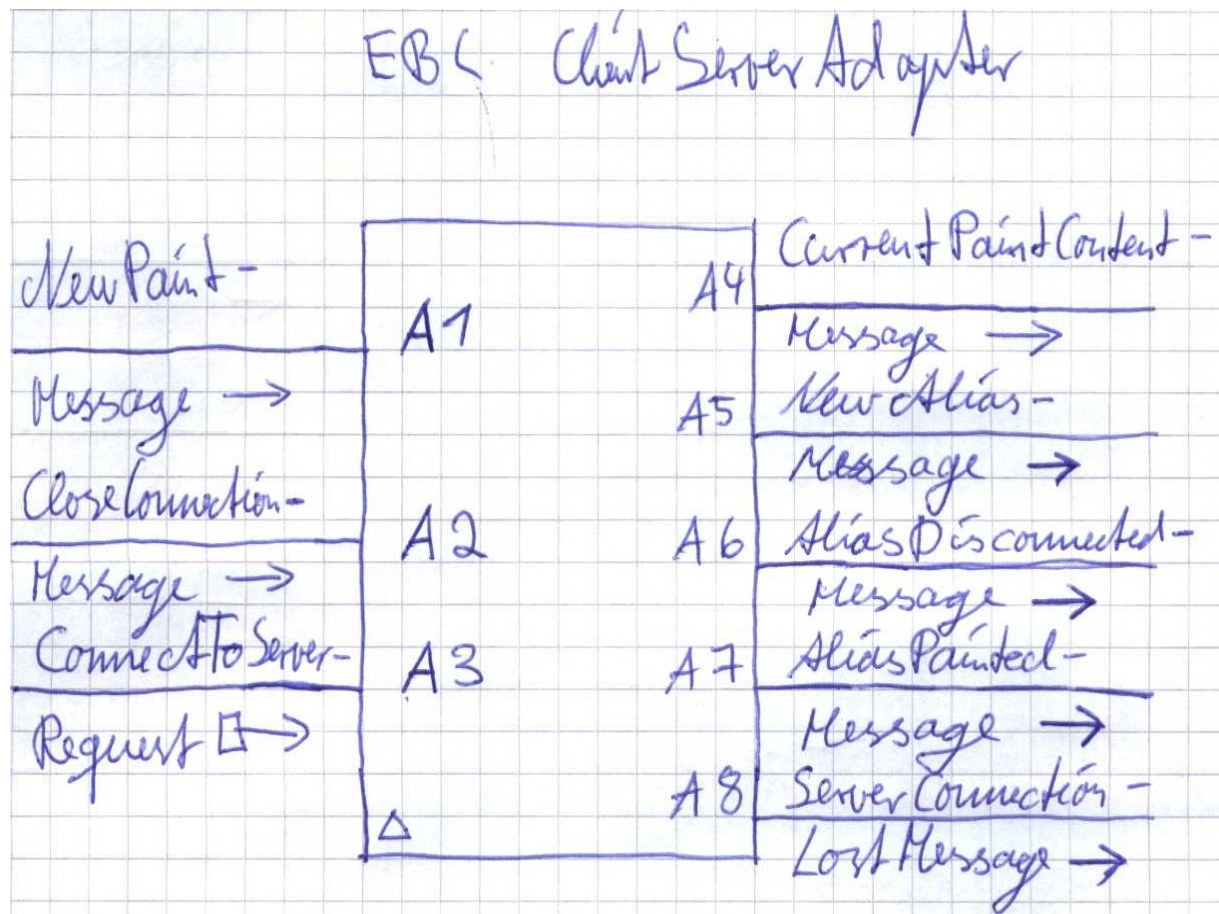
Die Aufgaben

- Überwachen der Verbindung um Nachrichten zu Empfangen
- Senden von Nachrichten über die Verbindung
- Umwandeln von Nachrichten nach XML und umgedreht
- Kodieren von XML in Bytes und Dekodieren der Bytes in XML

haben wir bereits beim Entwurf des PaintTogetherServer in eine eigene EBC, dem PaintTogetherCommunicater ausgelagert. Diese EBC werden wir im Adapter für diese Aufgaben verwenden. Zu entwerfen bleibt noch die Weiterleitung der zu sendenden Nachrichten von der Kernanwendung (Malen, Beenden, Starte Verbindung) an den Communicater sowie das

Informieren der Kernanwendung über eingegangene Nachrichten (Neuer Beteiligter, Beteiligter getrennt, Es wurde gemalt, Malbereichsinhalt und Beteiligtenliste).

Da der Adapter die PaintTogetherCommunicater-EBC verwendet, handelt es sich um eine Platine. Die noch nicht durch den PaintTogetherCommunicater behandelten Aufgaben übernimmt also min. eine zusätzliche EBC. Zuerst die Adapter-EBC von außen gesehen mit den konkreten In- und Outputs:



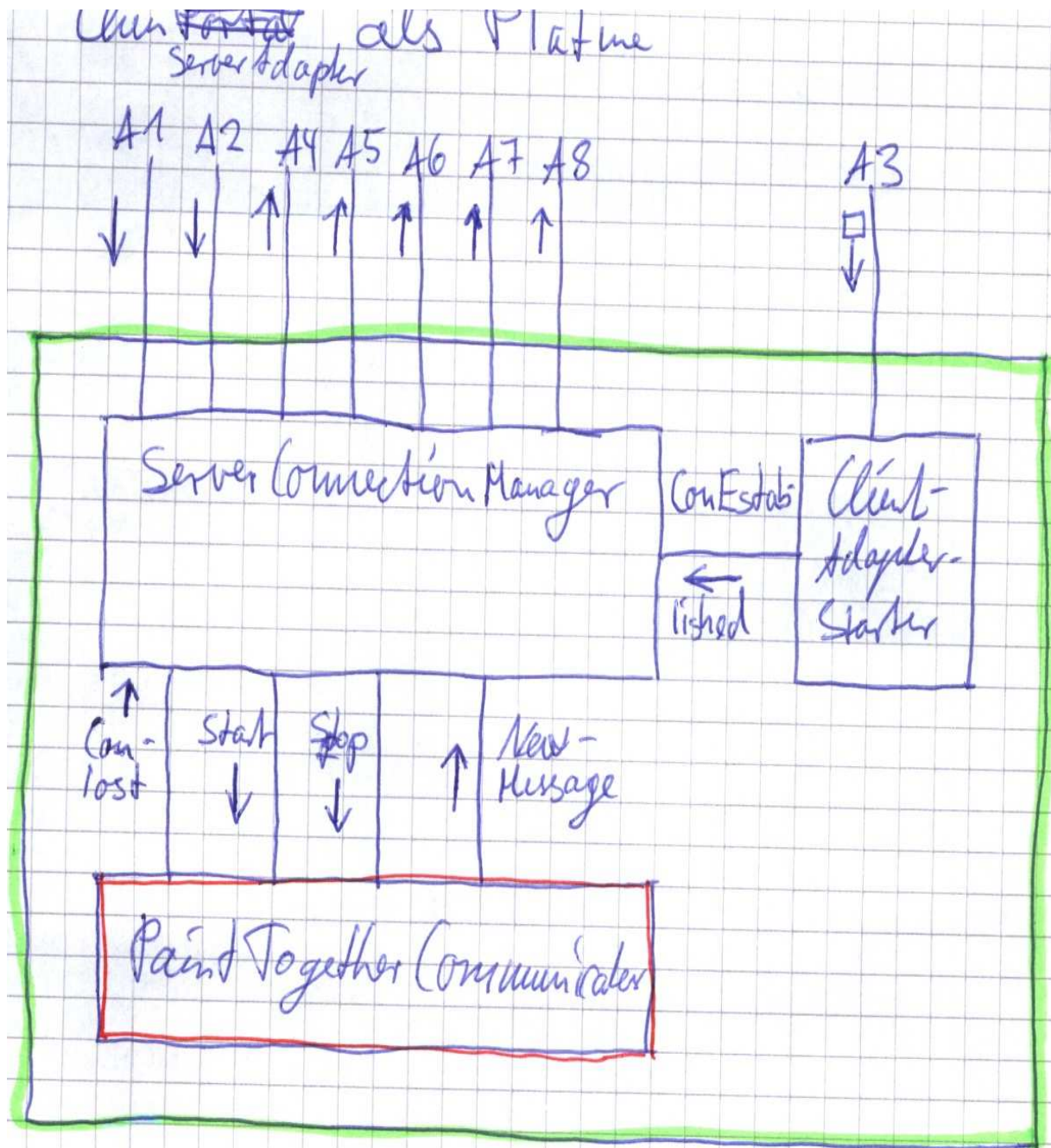
Die Nachricht ConnectToServerRequest beinhaltet den Servernamen oder IP und den Port des PaintTogetherServers sowie Farbe und Alias des Anwenders. Diese Nachricht besitzt eine Resulteigenschaft (hier Text), damit über das Ergebnis des Verbindungsaufbaus informiert werden kann.

Anmerkung:

Bei genauer Betrachtung entsprechen die drei Inputpins (A1-A3) exakt den drei einfachen Outputpins des ServerClientAdapters aus dem PaintTogetherServer.

Wenn das Portal über den Verbindungsverlust zum Server informiert wird, dann muss der Anwender darüber informiert werden. Die GUI soll dann zwar noch bedienbar sein, die Anwendung verhält sich dann aber wie eine einfache Standalone-Anwendung. Der Anwender kann noch malen, ein Austausch mit dem Server und somit mit anderen Beteiligten findet aber nicht mehr statt.

Die EBC als Platine:



Anmerkung zur Nachrichtenbenennung bei EBCs:

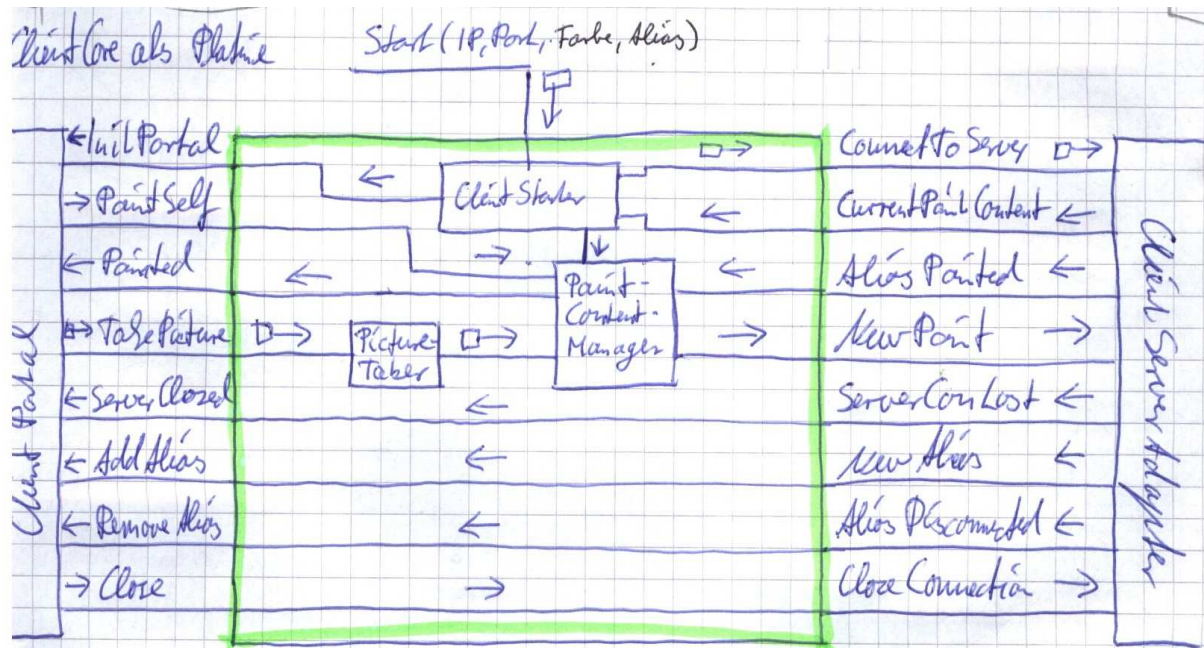
Viele hier und früher im Entwurf benannte Nachrichten besitzen den gleichen Inhalt. Bei den EBCs ist aber zwischen einem In- und Outputpin genau ein Nachrichtentyp zu definieren. Wenn ich zwischen dem Adapter und der Kernanwendung zum Beispiel den Alias und die Farbe für einen neuen Beteiligten übermittle, dann ist das ein anderes Nachrichtenobjekt als die Benachrichtigung des Portals aus der Kernanwendung über den neuen Beteiligten, wo auch die Farbe und der Alias übertragen wird. Aus diesem Grund habe ich versucht für jeden Nachrichtentyp einen anderen Namen zu vergeben. Später unterscheiden sich die Nachrichtentypen zusätzlich noch durch den Namespace. Doch soll das mehrfache Verwenden ein und desselben Namens vermieden werden, da es sonst zu Verwechslungen kommen kann. Warum macht man das? Dieses Vorgehen ermöglicht später eine einfachere Evolvierbarkeit der Software. Wenn zum Beispiel zwischen Adapter und Kernanwendung noch die IP des Clients übergeben werden soll, dann bekommt das Portal immer noch nur Alias und Farbe.

Das Problem mit der gleichnamigen Benennung habe ich auch bei der Benennung der EBCs versucht zu umgehen. Hier haben wir im PaintTogetherServer zum Beispiel die ConnectionManager-EBC auf der Platine ServerClientAdapter. Bei der Platine ClientSeverAdapter im PaintTogetherClient habe ich jetzt den Namen ServerConnectionManager gewählt. Nach außen hin ist später keine der beiden EBCs (ConnectionManager, ServerConnectionManger) sichtbar und somit hätte ich auch beide "ConnectionManager" benennen können. Ich habe mich aber dagegen entschieden, da das zu Verwirrungen führen könnte.

3.5.3 ClientCore

Bei der ClientCore-EBC müssen jetzt alle In- und Outputs des Adapters und des Portals bedient werden. Das sind jeweils acht Pins. Unsere EBC erhält neben diesen 16 Pins auch einen zusätzlichen Inputpin, einen Startpin. Damit entspricht das Design dem des PaintTopgetherServers, wo die ServerCore-EBC ebenfalls über einen Start-Input-Pin initialisiert wird.

Die EBC als Platine:



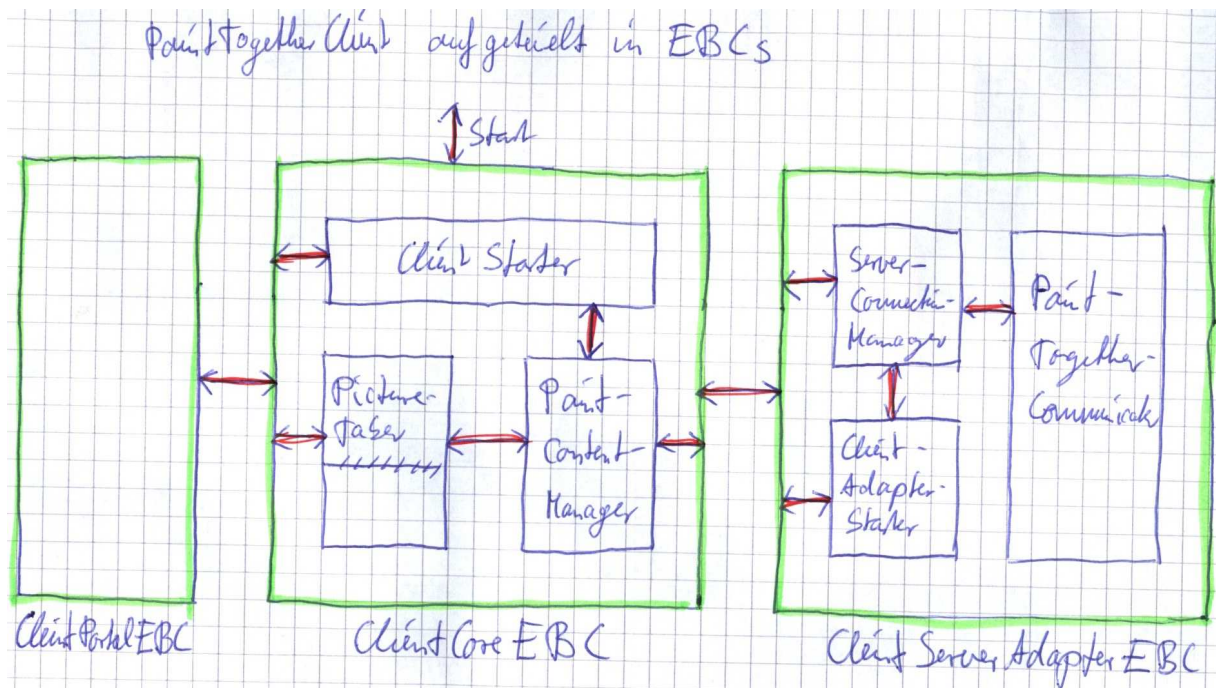
Die ClientCore-EBC unterteilt sich jetzt in die drei EBCs ClientStarter, PictureTaker und PaintContentManager.

Die Aufgabe des ClientStarters ist die Initialisierung des Adapters sowie des Portals. Außerdem initialisiert der ClientStarter auch die interne EBC "PaintContentManager" mit dem Startmalinhalt inkl. der Malfeldgröße.

Der PaintContentManager ist für die Verwaltung des Malbereichs am Client zuständig, damit der PictureTaker den aktuellen Malinhalt ermitteln kann.

Die PictureTaker-EBC soll nur das Speichern des aktuellen Malinhaltes unter dem angegebenen Dateinamen durchführen.

Für eine bessere Übersicht der im Client verwendeten EBCs stelle ich die Kommunikation der EBCs wie schon beim PaintTogetherServer in vereinfachter Form da:



4 Zusammenfassung

Der Entwurf aller EBCs und somit der gesamten Architektur der Anwendung PaintTogether, bestehend aus den drei einzelnen Anwendungen Server, Client, StartSelector sowie der EBC "PaintTogetherCommunicater" ist jetzt abgeschlossen. Der Entwurf der Oberflächen ist ebenfalls vollständig. Bei dem Entwurf des Servers wurde auch die Art der Socket-Verbindung beschrieben. Jetzt gilt es in der nächsten Phase, der Implementierung, dem Entwurf entsprechend die Software zu verwirklichen.