Dokumentation

Projekt: PublicScreen

Überblicksdokumentation

Dokumentversion 1.0

Erstversion:	23.06.06
Letzte Änderung:	18.07.06

Author: Technischer Rahmen

Inhaltsverzeichnis

1. Das PublicScreen-Projekt

Das PublicScreen-Projekt entstand in der Vorlesung „Programmentwicklung für mobile Geräte“ im Sommersemester 2006 an der Fachhochschule Augsburg in Kooperation zwischen den Fachbereichen Informatik und Gestaltung.

Das Ziel dieser Veranstaltung war die Entwicklung eines Systems, mit dem Spiele, Informationen und multimediale Inhalte per Handy auf einem großen, öffentlichen Bildschirm angesteuert werden können. Dieser große Bildschirm wird voraussichtlich im neuen Gebäude der Fachhochschule in der Schülestraße im Eingangsbereich vor der Mensa aufgestellt.

Damit für die Benutzer des Systems keine Gebühren anfallen, erfolgt die Verbindung zwischen Handy und dem PublicScreen per Bluetooth. Um sicherzustellen, dass die Anwendung auf möglichst vielen modernen Handys lauffähig ist, fiel die Entscheidung auf den Einsatz der Programmiersprache Java für mobile Geräte. Der PublicScreen-Server selbst ist ebenfalls zu einem großen Teil in Java realisiert, allerdings findet auch Flash und Python Verwendung.

2. Aufgabe des PublicScreen-Frameworks

Um mit mehr als 15 Projektteilnehmern effizient arbeiten zu können, wurde das Projekt in vier Aufgabenbereiche unterteilt und jeweils einer Gruppe von Studenten zugewiesen. Diese Aufgabenbereiche umfassten die Erstellung eines Informationssystems, der Entwicklung eines Spiels und dem Einbringen multimedialer Inhalte. Eine weitere Gruppe sollte sich damit befassen, einen gemeinsamen technischen Rahmen für die gerade aufgeführten Anwendungen

bereitzustellen. Das Konzept und die Realisierung dieses technischen Rahmens ist Inhalt der folgenden Kapitel.

Die Aufgabe der Gruppe „Technischer Rahmen“ beinhaltete folgende Bereiche:

Es musste eine Möglichkeit geschaffen werden, alle Anwendungen der anderen Gruppen in einem Projekt einzubinden. Es war nicht gewünscht, für jeden der vier Bereiche eine eigene Anwendung zu erstellen. Trotzdem sollte die Einbindung möglichst dynamisch erfolgen, um unabhängig vom Fortschritt der anderen Gruppen entwickeln, testen und zukünftige Erweiterungen einfach integrieren zu können.

Die Kommunikation war soweit zu vereinfachen, dass die Anwendungen direkt Nachrichten zwischen Handys und dem PublicScreen-Server versenden können, ohne Umwege über Bluetooth bzw. Socket-Programmierung selbst implementieren zu müssen. Außerdem sollte zusätzliche Funktionalität wie das Zurückliefern aller Handys in Reichweite des PublicScreen-Servers bereitgestellt werden.

Zudem war die Ansteuerung des Bildschirms eine Aufgabe des technischen Rahmens. Da in diesem Punkt aber Unstimmigkeiten mit den anderen Gruppen entstanden, wurde der Fokus von der Implementierung eines Verwaltungsmoduls zur Ansteuerung des Bildschirms hin zur Verwendung von Flash-Playern verlagert.

Ein System zur Authentifizierung von Benutzern wurde ebenfalls benötigt, damit die einzelnen Anwendungen auf deren Daten zugreifen können. Das wird z.B. von der „Informationssystem“-Gruppe verwendet um zu überprüfen ob ein Benutzer bereits seine Stimme für ein Voting abgegeben hat.

Neben der Programmierung selbst hatte die „Technische Rahmen“-Gruppe noch

zwei weitere Aufgaben. Die erste davon war die Koordinierung des Entwicklungsprozesses zwischen dem „Technischen Rahmen“ und den einzelnen Gruppen. Hier wurde neben viel geleistetem Support auch die Möglichkeit eingerichtet, das Versionsmanagement-System Subversion zu benutzen. Dadurch wurde die Entwicklung mit mehreren Leuten an einem Projekt über das Internet wesentlich vereinfacht und neue Versionen des Frameworks schnell für alle zugänglich. Im Detail erhielt jede Gruppe ein eigenes Repository zum Arbeiten. Jede Anwendung besteht aus drei Projekten, die wir bereits angelegt hatten: Einem gemeinsamen Teil (Shared), einer Client-Anwendung und einer Server-Anwendung. Im gemeinsamen Teil werden z.B. gemeinsame Schnittstellen und Basisklassen definiert.

Die zweite „Nicht-Programmier“-Aufgabe bestand in der Ausarbeitung von Rahmenbedingungen für das Gesamtprojekt. Hier wurde darauf eingegangen welche Bibliotheken verwendet werden durften, wie die Projektstruktur der einzelnen Gruppen aussehen sollte, sowie Einschränkungen bezüglich der Verwendung von Threads und Exceptionhandling definiert.

Hierbei ist natürlich die Client-Anwendung besonders hervorzuheben. Hier muss der User immer das letzte Wort behalten. Ein PlugIn kann nicht selbst entscheiden, ob Verbindung zum Server hergestellt werden oder die Anwendung heruntergefahren werden darf. Ein zentraler Kontrollpunkt muss hier die letzte Entscheidung behalten.

3. Nachrichtenübermittlung

In diesem Abschnitt wird erklärt, wie die Übermittlung von Nachrichten zwischen den Handys und dem PublicScreen-Server realisiert ist. Das Prinzip orientiert sich an einem Schichtenmodell, in dem die oberen Schichten auf den unteren aufbauen.

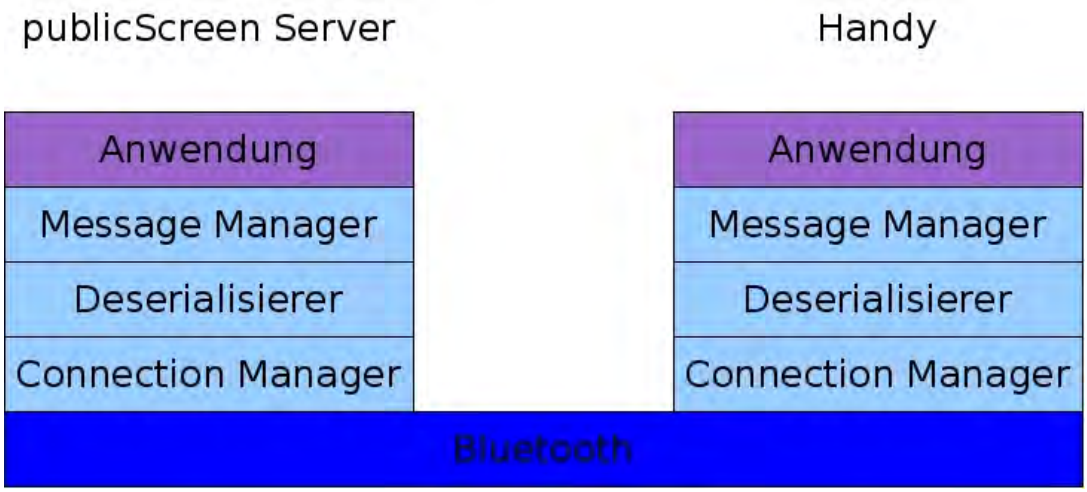

Abbildung 1: Vereinfachtes Modell des Kommunikationsprinzips

Die oben abgebildete Skizze zeigt ein vereinfachtes Modell des Kommunikationsprinzips. Zwischen dem Connection Manager und der Bluetooth-Schicht befinden sich noch weitere Komponenten, deren Erklärung an dieser Stelle aber nicht notwendig ist. Zu diesem Thema bietet sich die Dokumentation über den Connection Manager im Anhang dieses Dokuments an.

Wenn Daten zwischen einer Anwendung auf dem Handy an eine Anwendung auf dem PublicScreen-Server übertragen werden soll, wird dafür eine Nachricht (Message) erstellt und an den gewünschten Endpunkt (Anwendung auf dem

Server) adressiert. Diese Nachricht wird dann vom Message-Manager entgegen genommen, in einen Byte-Strom verpackt und über den Connection Manager an den PublicScreen-Server gesendet. Der Connection Manager auf Serverseite nimmt diesen Byte-Strom entgegen und leitet ihn an den Message-Manager weiter. Dieser wandelt den erhaltenen Byte-Strom über den Deserialization-Manager wieder in ein Message-Objekt um. Anschließend wird dieses Objekt an die adressierte Komponente weitergeleitet.

Der Deserialization-Manager verwendet Deserialisierer, mit denen er aus einem Byte-Strom Nachrichten extrahieren kann. Diese müssen bei ihm zu Anfang des Programms registriert werden. Die Serialisierung erfolgt über die Serialize-Methode des Message-Objekts. Diese Methode wird in dem Serializable-Interface definiert.

Wichtig ist noch, dass Komponenten nicht nur an entfernte Komponenten Messages senden können. Auch an lokale Komponenten können Nachrichten übermittelt und so Kommunikation durchgeführt werden.

Auch zum Thema Messaging und Deserialisierung ist im Anhang eine Dokumentation vorhanden, in der näher auf Details eingegangen wird.

4. Einbindung anderer Anwendungen

Wichtigste Anforderung im PublicScreen-Framework ist die Erweiterbarkeit und Einbindung von Anwendungen. Dies wird vor allem durch die Implementierung verschiedener Interfaces und die Registrierung bei den Kernkomponenten des Frameworks erreicht:

Zunächst müssen die Anwendungen einer gewissen Struktur genügen. Dies bedeutet einfach, dass ihr Einsprungpunkt eine Klasse ist, die von den Klassen

ClientPlugIn oder ServerPlugIn abgeleitet ist. Dadurch werden bestimmte Funktionen vorab definiert und einige bereits implementiert. Eine Anwendung kann natürlich auch aus mehreren PlugIns bestehen. Wichtige Funktionen im PlugIn wären die runPlugIn()- und die pausePlugIn()-Methode, welche bereits dem Namen nach die Verarbeitungsroutinen des PlugIns starten oder pausieren sollen.

Wozu diese Struktur? Die PlugIns der Anwendungen werden beim PlugIn-Manager (PIM) registriert. Nachdem alle Vorarbeit des Frameworks getan wurde, werden alle registrierten PlugIns durch den Aufruf der initializePlugIn()-Methode des jeweiligen PlugIns initialisiert. Dies beinhaltet so etwas wie z.B. die Abfrage, ob alle Vorbedingungen stimmen (Datenbank verfügbar?), oder das Laden von Graphiken. Schlägt die Initialisierung fehl, ist das PlugIn nicht verfügbar. Anschließend ist das Verfahren auf Server- und Client-Seite unterschiedlich:

Nach der Initialisierung der PlugIns wird auf Client-Seite das Basis-PlugIn gestartet. Dieses präsentiert dem User ein Menü, mit dem er verschiedene Unter-PlugIns anstarten kann. Das Starten erfolgt im PIM über die Methode runPlugIn() des ausgewählten PlugIns. Eine (aus der Historie entstandene) Besonderheit: Nach dem Beenden der Methode wird sie erneut aufgerufen, ein anderes PlugIn wird nur angestartet, wenn der PIM die Aufforderung dazu erhält.

Auf Server-Seite werden nacheinander alle registrierten PlugIns in einem jeweils eigenen Thread gestartet, da diese ja jeweils Server für ihre Anwendungen bilden.

PlugIns unterbrechen ihre Arbeit über den Aufruf der pausePlugIn()- bzw. stopPlugIn()-Methoden. Beim Pausieren soll die Anwendung ihre belegten Ressourcen behalten und auf ein Wiederanfahren warten, beim Stoppen werden

diese freigegeben und die Anwendung endgültig beendet. Nach dem Stop soll das PlugIn über die resetPlugIn()-Methode wieder in den Ausgangszustand (nach Initialisierung) versetzt werden.

Diese Mechanismen wurden aufgrund der knappen Zeit und dem Anspruch auf schnelle Umsetzung nur sporadisch umgesetzt, siehe dazu das Kapitel Verbesserungsvorschläge.

Auch die Kommunikation spielt eine wichtige Rolle. Wie können Anwendungen im Nachhinein ins Kommunikations-Netzwerk eingebunden werden? Dies erfolgt relativ einfach über eine Registrierung beim Message-Manager (MM). Dazu muss die kommunizierende Komponente das Interface AddressableComponent implementieren, über das mehrere Funktionen definiert werden. So z.B. die getEndpoint()-Methode, mit der der Kommunikations-Endpunkt der Komponente ermittelt wird oder die receive()-Methode, über die eine Message an die Komponente übergeben wird. Die Komponente wird dann in der Konfigurationsphase registriert.

Dieses Verfahren setzt sich fort, z.B. beim Recordstore-Manager auf Client-Seite oder dem Flash-Manager auf Server-Seite. Die Anwendungen registrieren beteiligte Objekte und werden á la Hollywood „zurückgerufen“.

Diese Registrierung erfolgt in sogenannten Konfigurations-Klassen, wie z.B. der TRConfiguration-Klasse. Dort werden die PlugIns beim MM und PIM registriert. Auch andere Vorarbeit, wie z.B. das Initialisieren des Recordstore-Managers, wird vollzogen.

Dieses Verfahren wurde von den anderen Projektgruppen ebenfalls durchgeführt, jegliche Initialisierung von Komponenten und Einbindung ins

Framework erfolgt in einer gruppenspezifischen Konfigurationsklasse. Dadurch können die Gesamtanwendungen „PublicScreen-Server“ und „PSI“ (Handy-Client) durch einfaches „Hintereinander-Aufrufen“ dieser Konfigurationsklassen erstellt werden.

5. Benutzersystem

Auch ein öffentlich zugängliches System wie der PublicScreen benötigt eine Benutzerverwaltung. Es können zwar auch Dienste ohne Benutzerdaten verwendet werden, allerdings erfordern manche Anwendungen die Möglichkeit zur benutzerabhängigen Datenablage. Ein Beispiel hierfür ist das Aquarium, in dem jeder registrierte Benutzer seinen eigenen Fisch anlegen und dessen Entwicklung weiter verfolgen kann.

Die Benutzerverwaltung konnte leider nicht komplett fertig gestellt werden, (fehlende Punkte sind im Kapitel „Verbesserungsvorschläge“ erläutert) trotzdem folgt hier eine Erklärung des ausgearbeiteten Konzepts.

Ein Benutzer, der sich registrieren möchte, legt auf einer Webseite einen Account an. Hierfür muss er einen eindeutigen Benutzernamen, ein Passwort und seine E-Mail-Adresse angeben.

Meldet sich der Benutzer zum ersten Mal mit seinem Handy am PublicScreen-Server an, wird in der Datenbank, die seine Benutzerdaten enthält, die Handy ID seines Handys mit dem Benutzeraccount verknüpft. Der Benutzer kann sich anschließend nur noch mit genau diesem Handy am System anmelden. Er kann aber bei einem neuen Handy diese Verknüpfung über die Webseite wieder aufheben.

Daraus ergeben sich zwei Konsequenzen: Ein Benutzer kann sich immer nur mit

einem Handy anmelden, zwei Benutzer auf dem selben Handy sind aber auch möglich.

Die Webseite sollte folgende Optionen bieten: Anlegen eines Benutzeraccounts, Einloggen, Rücksetzen des Passworts (Neues Passwort wird per E-Mail versendet) und Anzeigen von Benutzungshinweisen. Sollte der Benutzer auf der Webseite eingeloggt sein, sind ihm die Möglichkeiten zur Änderung seiner Daten, dem Rücksetzen der mit seinem Namen verknüpften Handy ID und dem Löschen seines Accounts zu gewähren.

Das PublicScreen-Framework sieht zur internen Verwaltung von Benutzern drei Listen vor. Es handelt sich dabei um die DIRL, die GUL und die PSUL.

Unter DIRL (**D**evices **I**n **R**ange **L**ist) versteht man eine Liste, in der alle sich in Empfangsreichweite des PublicScreen-Servers befindenden Handys eingetragen werden. Das schließt sowohl Handys, die am System angemeldet sind, als auch Handys mit aktiviertem Bluetooth in der Nähe ein.

In der GUL (**G**lobal **U**ser **L**ist) werden alle am System angemeldeten Benutzer erfasst. Im Gegensatz zur DIRL werden hier tatsächlich Benutzerdaten aufgelistet, während in der DIRL lediglich eine Liste mit Handys und dem zuletzt damit eingeloggten Benutzer verwaltet wird.

Die PSUL (**P**lugIn **S**pecific **U**ser **L**ist) ist eine Liste mit Listen. Jede dieser Listen enthält die Benutzer jeweils eines PlugIns. Die Vereinigung aller durch die PSUL verwalteten Listen entspricht der GUL.

6. Verbesserungsvorschläge

Natürlich gibt es trotz dem gewaltigen Umfangs des Frameworks noch viel Raum für Erweiterungen und Verbesserungen.

Es fehlt die Webseite zum Anlegen und Verwalten von Benutzerdaten sowie der Download-Möglichkeit für die Handy-Applikation. Evtl. ist diese Webseite im Rahmen einer anderen Veranstaltung wie Enterprise-Computing entwickelbar. Es kann auch darüber nachgedacht werden, die Applikation auf einem anderen Weg als per Web-Download auf die Handys der Anwender zu laden. Dies wird allerdings ein schwieriges Unterfangen, da verschiedene Handy-Hersteller unterschiedliche Verfahren für das Hochladen von Programmen auf ihre Geräte vorsehen. Bei Siemens-Handys reicht z.B. das Annehmen einer Datei über die Bluetooth oder Infrarot Schnittstelle während Nokia-Geräte einen speziellen Application-Installer voraussetzen.

Es wurde auch nie ausgetestet, wie viele Clients maximal am System angemeldet sein können. Prinzipiell bietet pythoncomm (siehe hierzu die Dokumentation des Connection Managers) zwar die Möglichkeit vor, beliebig viele Bluetooth Geräte anzusteuern und deren Daten an den Connection Manager weiterzuleiten, aber ein Problem könnte sich dennoch ergeben: Ein Handy kann nicht wissen, an welchem der Bluetooth-Adaper des Servers noch ein freier Slot zur Verfügung steht. Der Optimalfall würde eintreten, wenn ein Adapter mit erreichter maximaler Verbindungsanzahl vom Handy überhaupt nicht mehr gefunden werden kann. Dies konnte von uns aber mangels sieben bluetooth-fähiger Handys nicht ausgetestet werden.

Eine Lastverteilung über ein Routing System zu realisieren, scheitert unseres Erachtens an dem Umstand, dass geöffnete Bluetooth-Ports analog zu TCP-Ports

nicht an einen (Netzwerk- bzw. Bluetooth-) Adapter gebunden, sondern von jedem Adapter aus angesprochen werden können. Eine Lenkung der Verbindungsanfrage ist aus diesem Grund nur schlecht möglich.

Eine weitere Schwachstelle in unserem Kommunikationssystem ist der Umstand, dass die IDs aller vom Server verwendete Bluetooth-Geräte im Handy hinterlegt sein müssen. Es erfolgt aus Sicherheits- und Performancegründen keine dynamische Suche nach Bluetooth-Geräten, welche den PublicScreen-Service anbieten. Sollten alle Bluetooth-Adapter im PublicScreen-Server durch neue Hardware ersetzt werden, kann kein Handy diesen mehr finden. Hier wäre ein Erweiterung sinnvoll, in der beim ersten Aufruf der Applikation auf dem Handy nach Geräten mit dem PublicScreen-Service gesucht und die Ergebnisse der Suche im Recordstore abgelegt werden. Bei zukünftigen Verbindungsversuchen könnte dann solange auf die gespeicherten Ergebnisse im Recordstore zurückgegriffen werden, bis der Austausch der Hardware des Servers eine neue Suche erforderlich macht.

Auch die Einführung eines „Heartbeat“-Algorithmus steht noch aus. Damit ist das zyklische Senden einer Nachricht an das Basis-PlugIn verbundener Handys gemeint. Das Handy muss bei Erhalt dieser Nachricht innerhalb eines definierten Zeitintervalles eine Bestätigung an den Server zurücksenden, ansonsten wird die Verbindung als beendet betrachtet. Mit diesem Vorgehen können unsauber getrennte Verbindungen identifiziert werden. (Zum Beispiel wenn ein Handy nicht mehr innerhalb der Bluetooth-Empfangsreichweite ist)

Zudem wäre eine Vereinheitlichung im Bereich Datenbanken sehr vorteilhaft. Durch die relativ späte Einbindung einer Datenbank durch den Technischen Rahmen haben die anderen Arbeitsgruppen eigene Lösungen gesucht und dabei MySQL verwendet. Die Benutzerverwaltung im Framework selbst läuft allerdings auf der Datenbank Hypersonic. Durch die Vereinheitlichung wäre es dann nicht mehr notwendig beide Datenbanken installieren bzw. ausführen zu müssen.

Es sollte zudem noch die PSUL im Framework fertig implementiert werden. Diese verwaltet für jede Anwendung eine Liste mit aktuell dazu verbundenen Benutzern. Aus Zeitgründen konnte diese Liste von uns leider nicht mehr vollständig umgesetzt werden.

Eine weitere Verbesserung könnte erzielt werden, indem man Konfigurationsaspekte (z.B. Adresse des Servers) aus dem Java Quellcode in Property- oder INI-Files auslagert. Dadurch würden kleine Änderungen auch von jemand durchführbar, der keine Kenntnisse des Java-Quellcodes hat. Der einzige Programmteil, der bisher mit Konfigurationsdateien arbeitet ist unser COMM-Modul pytoncomm. Die Konfiguration dieses Programms muss an die IDs der verwendeten Bluetooth-Hardware und der IP-Adresse des PublicScreen-Servers angepasst werden. Da sich hier öfter Änderungen ergeben können, war die Auslagerung in eine extra Datei sehr wichtig.

Auch an der graphischen Gestaltung des Hauptmenüs der Handy-Applikation kann durchaus noch gearbeitet werden. Hier werden die Menüpunkte momentan nur als Klartext angezeigt, es wären aber zumindest ein ansprechendes Hintergrundbild sowie Icons für die einzelnen Punkte wünschenswert. Ebenso könnte die Textdarstellung des Verbindungsvorgangs durch eine kleine Animation ersetzt werden.

Dies geht einher mit der bisher noch etwas mangelhaft umgesetzten PlugIn-Struktur. Momentan findet man im Quellcode der PlugIns reihenweise leere Funktionen, die nur angefügt wurden um keine Fehler zu produzieren. Da diese Funktionen im Endeffekt immer die gleichen Aufgaben haben und diese Aufgabe in der Regel auf die gleiche Art und Weise vollzogen werden kann, bietet sich hier ein stärkerer Ausbau der PlugIn-Basisklassen an. Diese sollte einen großen Teil des Laufzeit-Managements übernehmen und durch Schablonen-Methoden (vgl. Design-Patterns) von den Kind-Klassen erweitert werden. Dabei könnten für unterschiedliche Verarbeitungsdogmen Basisklassen vorbereitet werden, so z.B.

eine Basisklasse in der runPlugIn von den Kind-Klassen implementiert und nach dem Beenden verlassen wird oder eine Basisklasse, in der ein doRunPlugIn() zyklisch aufgerufen wird, bis das PlugIn beendet wird.

Des weiteren sollten manche Manager wie z.B. der Message-Manager überarbeitet werden. Auf Server-Seite wurde versucht durch eine über-komplexe Struktur eine Art Pipeline zu implementieren. Diese Struktur resultierte aus dem Versuch, einen Großteil des Codes auf Client- und Server-Seite zu verwenden. Leider entsteht auf dem Handy beim Empfangen größerer Messages (größer als 350 Bytes) eine Heap-Exception. Dank Versionierungssystems konnte rechtzeitig vor der Präsentation ein ältere und stabilere Version wiederhergestellt und verwendet werden. Aber grundsätzlich ist eine Pipeline-Struktur wünschenswert. Serialisierung und Deserialsierung vor allem komplexer Inhalte sollte nebenläufig zur Zustellung der Message erfolgen.

Des weiteren werden auf Server-Seite Messages an die Komponente nur der Reihe nach zugestellt. Das heißt, es wird mit der nächsten Nachricht solange gewartet bis die vorherige abgearbeitet ist. Dies wurde eingeführt um einfachere Programmierung der einzelnen Gruppen zu ermöglichen, wird aber im laufenden Betrieb mit mehreren Handy-Clients zu unerwünschten Blockaden führen.

Der letzte große Verbesserungsvorschlag beinhaltet das Hinzufügen neuer Inhalte. Durch das Konzept der festen Aufteilung des Bildschirms in die drei Bereiche Aquarium, Informationssystem und Spiel ist hier allerdings eine nicht einfach zu durchbrechende Grenze gesetzt. An dieser Stelle sollte eventuell das gesamte Konzept der Bildschirmaufteilung nochmals überdacht werden.

Eine bisher allerdings komplett ungenutzte Möglichkeit neue Inhalte in das PublicScreen-Framework zu integrieren, besteht im Bereich von Location Services. Das PublicScreen-Framework erlaubt das Aufstellen von Empfangsstationen an beliebigen Orten, an denen der Zugang ins FH Netzwerk

möglich ist. Die Anwendungen könnten dann auf die Information zugreifen, zu welcher Empfangsstation (Stichwort „PeerID“, siehe Dokumentation des Connection Managers) sich ein Handy verbunden hat und daran entscheiden, welche Dienste dem Benutzer an diesem Ort zur Verfügung stehen. Hier liegt der Fokus der Dienste dann nicht mehr auf der Ausgabe auf dem öffentlichen Bildschirm, sondern auf der Darstellung auf dem Handy-Display des Benutzers.

7. Verweise

Neben dieser Überblicks-Dokumentation wurden vom „Technischen Rahmen“ noch einige weitere Dokumentationen erstellt, die detaillierter auf einzelne Mechanismen im Framework eingehen.

Diese Dokumentationen befinden sich im Anhang dieses Dokuments und haben folgende Inhalte:

- Funktionsweise des Connection Managers
- Erstellen und Einbinden eigener PlugIns
- Funktionsweise Messaging
- Verwendung des Subversion-Servers
- Konfiguration des Systems
- Verwendung des Recordstores
- Funktionsweise des Screen Managers (dieser befindet sich nach wie vor zu Debug- und Management-Zwecken im System, seine Hauptaufgabe wurde allerdings durch die Verwndung von Flash-Playern ersetzt)
- JavaDoc

8. Hinweise

Da die Gruppe „Technischer Rahmen“ nahezu alle Teilbereiche gemeinsam bearbeitet hat und eine klare Trennung in Aufgabengebiete kaum möglich ist, bitten wir um eine einheitliche Benotung der gesamten Gruppe.

Auf der beiliegenden CD befindet sich ein aktuelles Abbild des SVN-Servers.

Marcel Kieser

Clemens Rückle

Harald Manske