**Ergebnisprotokoll vom Teammeeting am 20.05.2015**

**Anwesend:** Arno Däuper, Niklas Kröger, Inga Quatuor, Sebastian Schwanewilms, Jan Zimmer

**Protokollantin:** Inga Quatuor

**Inhalt:**

1. Dokumentation
2. Ergänzungen / Änderungen der Spezifikation
   1. Clients
   2. Messages & Timestamps
   3. Netzwerkschicht
   4. Präsentation
   5. Frage stellen
   6. Gestensteuerung
   7. Audioprocessing
   8. Sonstiges
3. Arbeitspakete / Aufgabenverteilung

**1. Dokumentation**

Den Abschlussbericht wollen wir auf Deutsch schreiben. Zwecks Einheitlichkeit sollen daher auch die einzelnen Bestandteile der Dokumentation – Use Cases, Arbeitspakete, Konzeptbeschreibungen der Systemkomponenten etc. – auf Deutsch sein.

Der Code selbst hingegen wird in englischer Sprache gehalten. Das betrifft Klassen- und Funktionsnamen, Kommentare usw. Klassendiagramme sind entsprechend auch in Englisch.

Use Cases: Die aktuelle Version 2.0 ist um den UseCase „Ausgabemedium einrichten“ zu erweitern. Die unten gelisteten Änderungen müssen teilweise eingepflegt werden.

**2. Ergänzungen / Änderungen der Spezifikation**

Die beim letzten Meeting vereinbarte Projektspezifikation wird wie folgt geändert bzw. ergänzt.

**2.1 Clients**

* MasterClient und ListenerClient sind getrennte Applikationen.
* Der MasterClient muss sich bei der Anmeldung mit Passwort authentifizieren.
* OutputClient: Der OutputClient wird nicht vom MasterClient ausgewählt und beim Server hinterlegt, sondern jeder Client kann sich selbst jederzeit zum OutputClient machen.
* (Weitere Details siehe Abschnitt 2.4 und 2.5)

**2.2 Messages & Timestamps**

* Sämtliche Kommunikation zwischen Server und Clients erfolgt in einem festen XML-Format.
* Jede Message enthält auf jeden Fall Sender, Empfänger und einen Timestamp (vom Senders).
* Audiofiles erhalten (lokale) Timestamps vom Start- und Endzeitpunkt der Aufnahme.
* Alle Messages passieren den Server und werden dort geloggt.
* Erläuterung: Durch die Menge der Timestamps sollte es möglich sein, auch bei größeren Delays die zeitliche Struktur des Vortrags präzise zu bestimmen. Ziel ist, dass die Audioaufnahmen der unterschiedlichen Devices später präzise zusammengebaut werden können (auch wenn das in unserem Projekt nicht vorgesehen ist).

**2.3 Netzwerkschicht**

* Ziel: Sender schickt Nachricht an Empfänger und bekommt eine Antwort, ob der Vorgang erfolgreich war.
* Hierzu soll die Netzwerkschicht ein entsprechendes Interface zur Verfügung stellen, welches die innere Logik (Datagramme, Sockets, etc.) nach außen verbirgt und z. B. die Funktionen init() und send() liefert.
* Damit die Übertragung von kurzen Kommandos nicht durch Datentransfer behindert wird, sollen immer zwei Sockets geöffnet werden: einer für Kommandos und einer für Daten.
* Die Netzwerkschicht teilt dem Server mit, wenn ein Client disconnectet. (Details bzgl. der Aufgabenteilung zwischen Netzwerkkomponente und Server müssen hier ggfls. noch genauer besprochen werden.)

**2.4 Präsentation**

* Die gesamte Präsentation wird am Anfang an alle Clients verteilt (anstatt dass immer nur die aktuelle Folie übertragen wird).
* Während des Vortrags wird dann vom Server die anzuzeigende Folie an die Clients kommuniziert. (Hierbei wird die Seitenzahl geschickt und nicht nur das Kommando „vor“ oder „zurück“.)
* Die Steuerung der Präsentation ist auch während einer Frage aus dem Publikum möglich.

**2.5 Frage stellen**

* Jeder ListenerClient kann eine Redeanfrage stellen. Derselbe ListenerClient kann keine weitere Redeanfrage senden, solange seine aktuelle Anfrage nicht beantwortet wurde. (Implementierung clientseitig)
* Redeanfragen von verschiedenen ListenerClient werden gequeuet. Die Queue ist nicht begrenzt (außer durch die Anzahl der ListenerClients).
* Wenn eine Redeanfrage angenommen wird, startet nicht sofort die Audioaufnahme beim Fragesteller, sondern der ListenerClient hat vorher noch die Möglichkeit, den Frage-Vorgang abzubrechen.
* Nur der MasterClient kann die Frage (und damit die Audioaufnahme) beenden. (Anm.: Hier soll noch überprüft werden, ob diese Einschränkung tatsächlich notwendig ist.)
* Der MasterClient soll zusätzlich über folgende Funktionalität verfügen: a) Keine-Fragen-Modus (jederzeit an- oder abzuschalten), b) alle Redeanfragen in der Warteschlange auf einmal ablehnen

**2.6 Gestensteuerung**

* Die Gestensteuerung soll vom MasterClient aktiviert und deaktiviert werden können.
* Die Gestensteuerung läuft (wie das GUI) in einem eigenen Thread. Wenn die zugehörige Klasse instanziert wird, wird der Thread erzeugt. Falls Verwaltung des Threads notwendig ist, erfolgt das in dieser Klasse.
* Die Klasse soll nur einmalig instanziert werden. Das Aktivieren und Deaktivieren erfolgt dann über entsprechende Funktionen.
* Es soll nur ein Signaltyp an den Client geschickt werden. Welches Signal es ist (vor oder zurück), kann als Parameter übergeben werden.
* Wichtig: Im Destruktor muss die Kamera-Ressource freigegeben werden!

**2.7 Audioprocessing**

* Vergleichlich zur Gestensteuerung: Eigener Thread, einmalige Instanzierung, Mikrofon-Ressource muss freigegeben werden.

**2.8 Sonstiges**

* Anmerkung: Falls die Applikationen per X beendet werden, muss sichergestellt werden, dass aufgeräumt wird. Neben den üblichen Destruktoren kann man bei QNX auch eine Routine implementieren, die in diesem Fall ausgeführt wird.

**3. Arbeitspakete / Aufgabenverteilung**

neu:

* Schnittstelle zum Ausgabegerät (HDMI): Arno

noch nicht verteilt:

* MasterClient – Details (aufbauend auf Client-Basis)
* ListenerClient – Details (aufbauend auf Client-Basis)
* MasterClient und ListenerClient – GUI