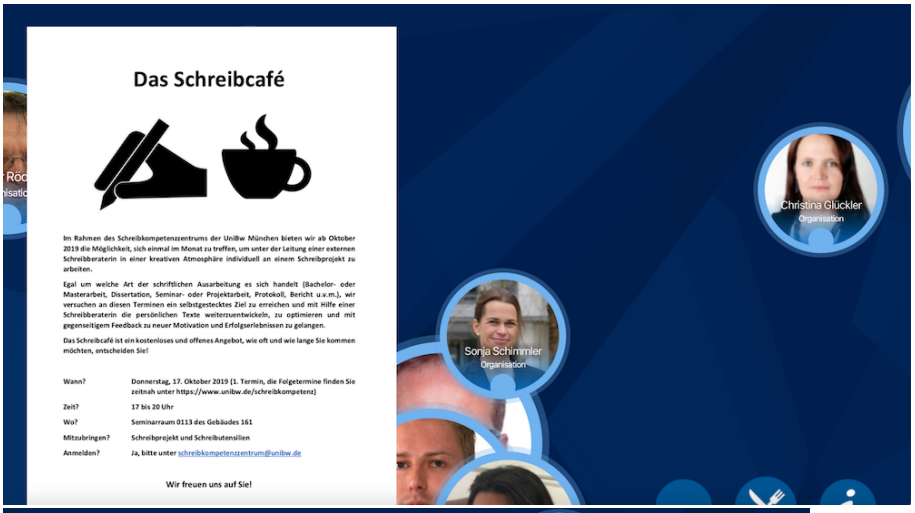
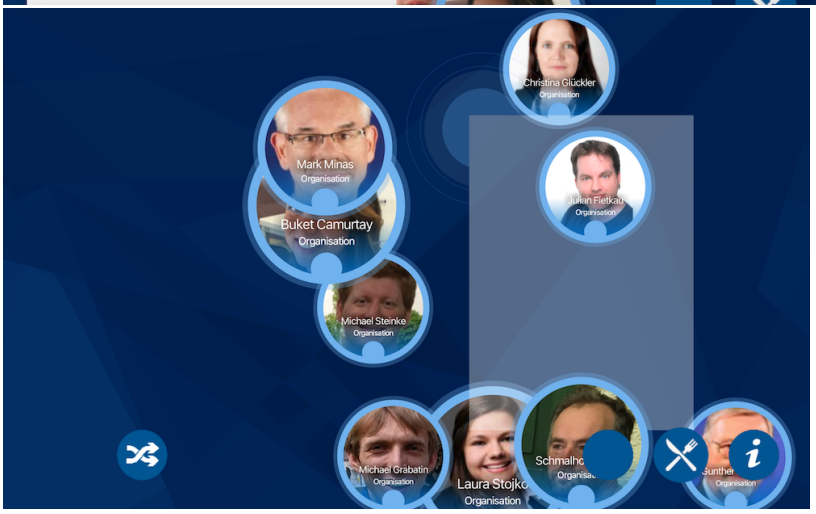
CMF 3.0

Dokumentation zum CommunityMirrorFramework 3.0 und damit realisierte CommunityMirror-Anwendundungen.





Das CMF 3.0 wurde 2019 als Nachfolger der Vorgängerversion CMF 2.0 realisiert. Ziele bei der Neuimplementierung waren die Beschränkung auf das Wesentliche und das Entschlacken des Sourcecodes. Als Basis für Eventbehandlung und Visualisierung/Animation wurde das JavaFX-Framework beibehalten.

Todos

- Picture-Content-Items: In Detail-View auch die Bildbeschreibung anzeigen
- Picture-Content-Items: Unterschiedliche Größen für landscape und portrait
- Bild für Icon zum Neu-Laden eines Teasers (dritter von rechts)
- Remote-Aktion bei Identifikation eines Benutzers
- In (Touch)Behaviors eine Möglichkeit zum Konfigurieren und Absetzen von Meldungen an LoggingFramework einbauen - und in StickyComponents etc. nutzen
- inPreview-Darstellung (FXML, Controller) für Items von Typ Organisation machen
- FreeMarker Templates für Detaildarstellung von Items vom Type content, person, Organisation machen, dazu auch CSS-File machen
- Testen von TouchEvents auf geeigneter Hardware (Test ob TouchHold funktioniert)

    - Check in wie weit JavaFX Multi-Touch und Multi-Displays (neue Hardware) unterstützt; dabei insbesondere: wie funktioniert Screen-übergreifendes Multi-User-Multi-Touch bei einem Großbildschirm bestehend aus einer Display-Matrix (mehrere Einzeldisplays
- ~~Flow: wenn ein FlowVisualItem angeklickt wird, dann dieses in den Vordergrund bringen (d.h. in andere Group einfügen oder in derselben Group nach vorne)~~
- Profiling: Klären, wo beim Aufruf von onUpdate in FlowView hin und wieder so viel Zeit verbraucht wird, dass die Animation stockt - und Beseitigung des Problems (durch Pre-Loading oder Auslagerung in einen anderen Thread)
- Bei Sticky Components noch mehr FXML und CSS (und JavaFX MVC) nutzen - wie bei VisualItems
- VisualItems: Konfigurierbare Wahl von FXML Dateinamen (in FXMLVisualItem.getFXMLFileName()) - d.h. Namen können abhängig vom Typ und von Metatags in Konfigurationsdatei festgelegt werden
- Graph-View umsetzen
    - Check 2D-Physik-Engines für JavaFX, z.B. Phys2D vgl. dazu u.a. auch https://books.google.de/books?id=D0Ml8WugPm8C&lpg=PA114&ots=BiHa9fhaCX&dq=Phys2D%20javafx&hl=de&pg=PA114#v=onepage&q=Phys2D%20javafx&f=false
- SelectionStrategyDefault weiter implementieren

    - Allgemein: Jeweils prüfen, ob es der erste Parameter ist - dann laden - oder ein weiterer - dann filtern
    - Vielleicht die Parameter automatisch umordnen, so dass das am besten passt? (Query Optimization ...)
- Möglichkeit zum Anmelden/Abmelden von Benutzern über Remote-Schnittstelle einbauen; Benutzer dann als Teil des Kontextes verfügbar - dazu Komponente in node.js bauen, die über Remote-Schnittstelle mit CMF3 spricht und Id-Daten über Bluetooth bekommt
- Build-Prozess: Klären, wie man mit JDK 11 und JavaFX 11 in Eclipse ein "Fat-JAR" erzeugen kann - also ein JAR, das alles enthält, was man zum Starten der CommunityMirror-App braucht - insbesondere die JavaFX-Bibliotheken/Modul - siehe z.B. http://rimdiary.com/javafx-on-jdk-11/ für einige Hinweise, wie das gehen könnte ...
- JavaDoc erzeugen (aus Eclipse)
- Logging-Framework (cmf3.log.LoggingFramework) weiter ausbauen (und auch von mehr Stellen aus aufrufen)
    - separater Logging-Server, Dashboard-Server, an den die Daten weitergegeben werden - siehe auch CommunityLogging
    - in View Funktionen einbauen, die Snapshots von Daten und Display erstellen und in Datei ablegen (als JSON?) - auslösen durch LoggingFramework-Klasse oder durch Remote-Aufruf
- (Separate) Komponenten bauen, mit denen Daten von Kinekt (Azure) und ähnlichen erfasst wird und an Logging-Server weitergegeben wird, optional auch über Remote-Schnittstelle an CMF3 (z.B. um auf Nutzer zu reagieren)
- Erste Version eines Dashboards - Anzeige des aktuellen Status von verschiedenen CMs, Anzeige von Fehlermeldungen, Anzeige aggregierter Nutzungsstatistiken, ...