Tourney
Entwurf

Softwarepraktikum
Wintersemester 2014/2015

Jonas Auer (2860992)
Fabian Biester (2859084)
Jan Tagscherer (2893134)

Universität Stuttgart
12.12.2014

Inhaltsverzeichnis

1 Einleitung

Dieser Entwurf bietet den Entwicklern von **Tourney** einen Überblick über die Architektur der Software. Dadurch wird ein Grundgerüst bereitgestellt, das die geplante Implementierung ermöglicht.

1.1 Zweck des Software-Systems

In diesem Entwurf wird die Architektur von **Tourney** beschrieben. Dabei handelt es sich um Software, die die bisher händisch erfolgte Organisation von Spielturnieren des Heidelberger Spieleverlags übernehmen und vereinfachen soll. Dazu soll das Programm die Organisatoren durch den kompletten Prozess der Turnierplanung begleiten und sie bei den typischen Aufgaben wie der Teilnehmerverwaltung und der eigentlichen Durchführung der Turniere unterstützen.

Besonderer Wert wird hierbei auf eine intuitive Bedienung gelegt, die mit möglichst wenig Aufwand zum Ziel führt.

1.2 Leserkreis des Entwurfs

Zum Leserkreis dieses Entwurfsdokuments gehören die folgenden Personengruppen:

- Die Entwickler der Software
- Die für die Wartung oder Weiterentwicklung zuständigen Software-Ingenieure
- Die Betreuer

1.3 Angewandte Entwicklungsprinzipien

Der Entwurf von **Tourney** findet im Top-Down-Prinzip statt. Hierzu wird die geforderte Gesamtfunktionalität des Systems betrachtet und dann zur Erfüllung dieser notwendige Aufgaben auf Komponenten und schließlich Klassen verteilt.

1.4 Überblick über den Entwurf

Das Entwurfsdokument wird in die folgenden Kapitel gegliedert:

Kapitel 1 gibt einen groben Überblick über das zu entwickelnde Softwaresystem und das hierzu erstellte Entwurfsdokument.

Kapitel 2 umfasst die Architektur, die die Software in überschaubare Komponenten gliedert. Diese werden dazu in einem Komponentendiagramm dargestellt.

Kapitel 3 beschreibt die in der Architektur definierten Komponenten je nach Bedarf über Klassendiagramme zur Schnittstelle, Sequenzdiagramme zum Protokoll und Zustands- und Aktivitätsdiagrammen zum Verhalten im Detail.

Kapitel 4 charakterisiert externe Schnittstellen und die verwendete Methodik zur persistenten Datenspeicherung.

Kapitel 5 enthält die Versionshistorie zu diesem Dokument.

2 Architektur des Systems

2.1 Überblick über die Architektur

Das geplante System wird in einer klassischen Architektur aus drei Schichten realisiert. Hierzu findet eine Gliederung in die drei Komponenten Benutzeroberfläche, Anwendungslogik und Datenhaltung statt. Da die Benutzeroberfläche über *JavaFX* erstellt wird ist diese standardmäßig in externe *.fxml*-Dateien ausgelagert. Die Anwendungslogik dient als Vermittler zwischen den anderen beiden Komponenten und reagiert auf Eingaben aus der Benutzeroberfläche mit den richtigen Operationen auf der Datenhaltungsschicht.

Bei der Architektur wird Wert auf eine geringe Kopplung zwischen und einen hohen Zusammenhalt in den Modulen gelegt. Dies wird unter anderem durch das Verwenden von *JavaFX-Bindings* gewährleistet. Hierbei werden Attribute von Klassen mit anderen Attributen oder Elementen der Benutzeroberfläche über eine Listener-Architektur gebunden und bei Bedarf neu berechnet oder aktualisiert.

2.2 Verwendete Entwurfsmuster

Bei der Entwicklung werden an einigen Stellen die folgenden bewährten Entwurfsmuster verwendet:

Observer

Ändern sich die Attribute von Objekten, so müssen häufig sowohl die Benutzeroberfläche aktualisiert als auch andere Attribute neu berechnet werden. In entgegengesetzter Richtung ändern einige Eingaben über die Benutzeroberfläche Attribute.

Dieser Zusammenhang wird über *JavaFX-Bindings* implementiert, wobei das Observer-Entwurfsmuster verwendet wird. Hierzu werden alle relevanten Attribute als *JavaFX-Properties* deklariert.

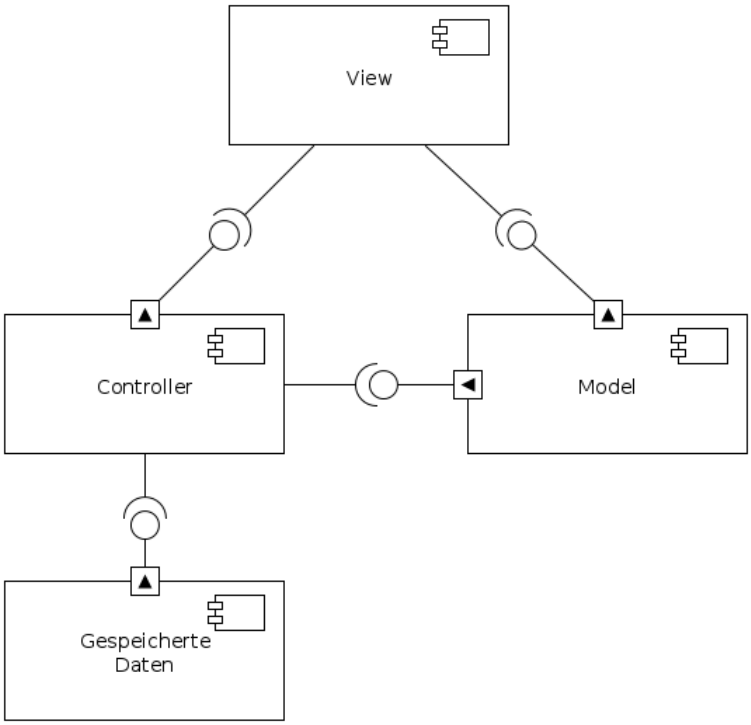
Strategy

Die Berechnung der Ergebnisse von Spielrunden findet immer abhängig von dem verwendeten Regelmodul statt. Dazu wird jedes mögliche Paarungsprinzip als Subklasse einer Strategie-Oberklasse hinzugefügt und kann austauschend verwendet werden. Diese Modularität trägt zur guten Erweiterbarkeit bei.

Singleton

Die Klasse *PreferencesManager* zur Verwaltung der Einstellungen darf pro laufendem System nur als eine Instanz vorhanden sein. Daher wird das Entwurfsmuster des Singletons verwendet.

2.3 Komponentendiagramm

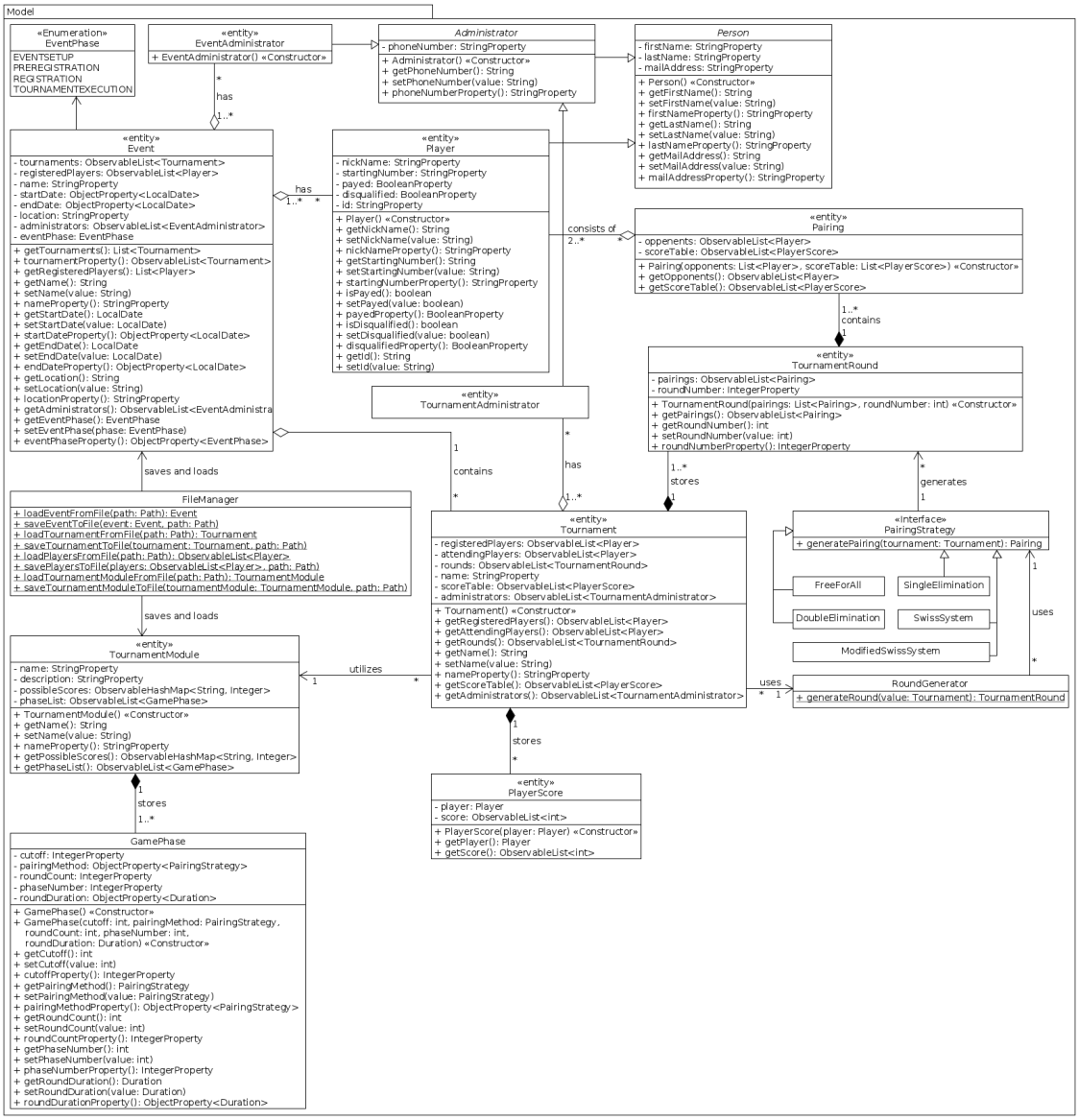


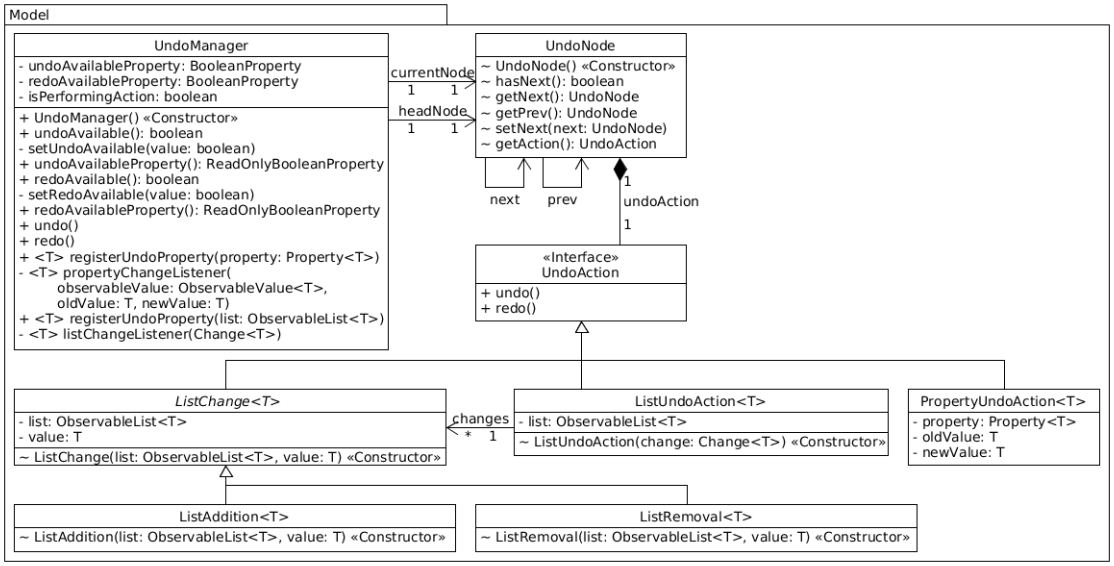
3 Komponenten

3.1 Model

Im folgenden wird die Schicht der Datenhaltung und Anwendungslogik beschrieben und strukturiert.

3.1.1 Klassendiagramme





3.1.2 Beschreibung

Zu Beginn steht die Klasse *Event*. Sie symbolisiert das echte Event und enthält die Spielerliste und die Turnierliste. In der Klasse *Tournament* läuft das Turnier ab, es werden die teilnehmenden *Player* und die bisherigen Runden und deren Ergebnisse gespeichert. In der Klasse *Spieler* werden die persönlichen Daten des Spieler wie Vorname, Nachname und E-Mail-Adresse gespeichert. Die Klassen sind reine Datenstrukturen, jegliche Logik wird in den *Controllern* ausgeführt.

Der *UndoManager* bietet eine einfach zu benutzende Schnittstelle, um es dem Benutzer zu ermöglichen, durchgeführte Aktionen wieder rückgängig zu machen. Dieses Verfahren lässt sich auf alle *JavaFX-Properties* und *JavaFX-Listen* anwenden, indem man sie durch *registerUndoProperty()* registriert.

Intern werden alle an den Properties gemachten Änderungen in einer verketteten Liste gespeichert, die aus *UndoNode*s aufgebaut ist.

Die ausgeführten Änderungen werden innerhalb der Nodes als konkrete Implementierung des *UndoAction* Interfaces abgelegt, das die beiden Methoden *undo()* und *redo()* bereitstellt. Die Implementierungen kümmern sich dann um die konkrete Ausführung der Rückgängig- und Wiederherstell-Aktionen.

Wenn an *JavaFX-Properties* Änderungen erfolgen, werden diese als *PropertyUndoAction* gespeichert. Der vorherige und der neue Wert werden gespeichert, um später rückgängig gemacht oder wiederhergestellt werden zu können. Änderungen an *JavaFX-Listen* sind von etwas komplizierterer Natur, da hier pro Change-Event mehrere Hinzufüge- und Entfernen-Aktionen zusammengefasst sind. Daher wird dieses Verhalten in einer *ListUndoAction* gespiegelt, die verschiedene *ListChange*s beinhaltet. Dabei entsprechen *ListAddition*s dem Hinzufügen von Werten zu einer Liste und *ListRemoval*s dem Entfernen von Werten von einer Liste.

3.2 View

Beim View handelt es sich um die eigentliche grafische Benutzeroberfläche, die über *JavaFX* implementiert wird.

3.2.1 Beschreibung

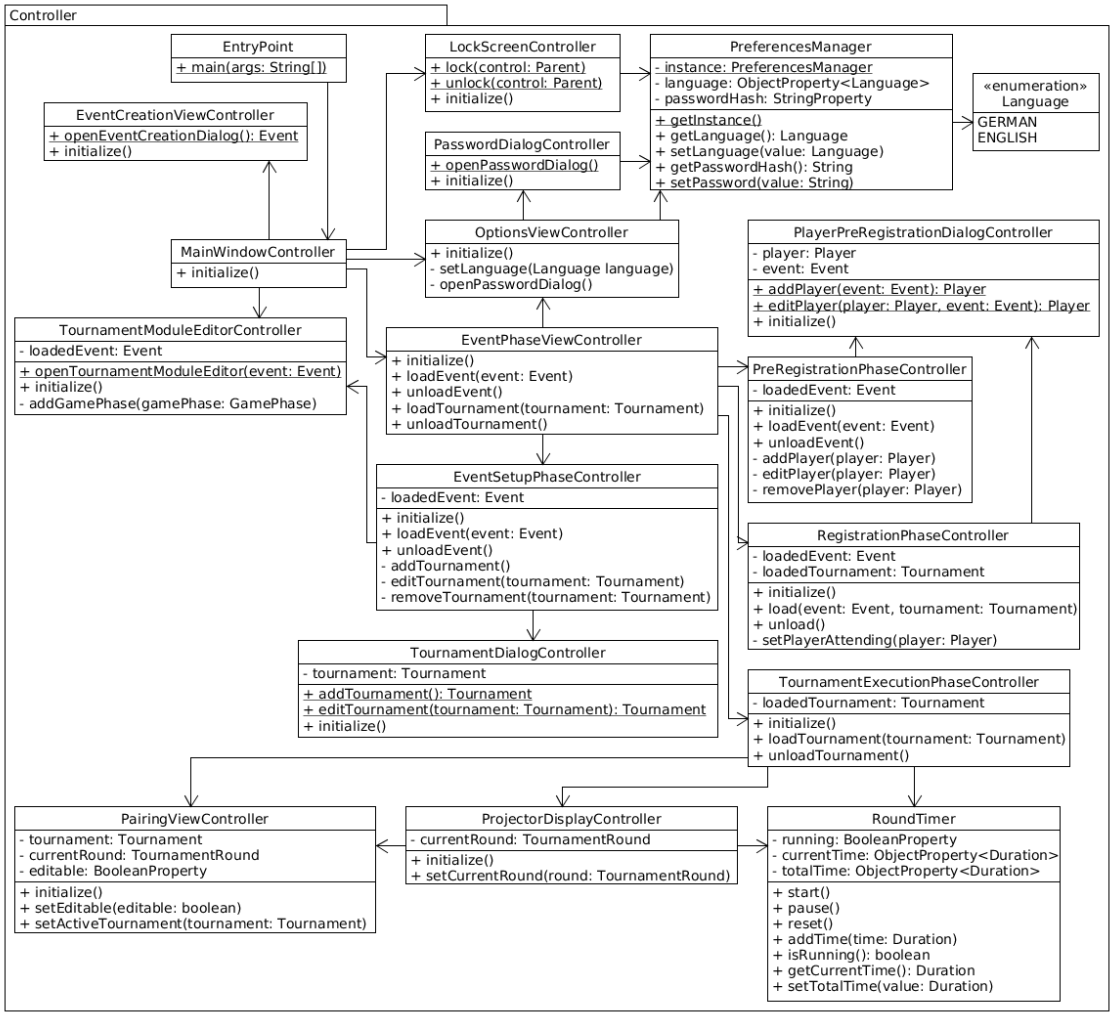
Die Benutzeroberfläche in *JavaFX* besteht aus jeweils einer *.fxml*-Datei für jede in der Kontrollstruktur beschriebene Controller-Klasse. Diese Dateien enthalten XML-strukturierte Daten, die die Oberfläche jeweils eines Fensters beschreiben und die Callback-Methode bei einer durch den Nutzer initiierten Aktion festlegen.

Durch diese Strukturierung ist eine komplette Kapselung der Benutzeroberfläche mit geringer Kopplung zu den anderen Komponenten gegeben.

3.3 Controller

Die Kontrollstruktur nimmt Eingaben von der Benutzeroberfläche entgegen und führt anhand dessen die passenden Operationen auf der Datenhaltungsschicht aus. Die Struktur wird im Folgenden beschrieben.

3.3.1 Klassendiagramm



3.3.2 Beschreibung

Die Controller in der Kontrollschicht spiegeln die Fenster der Anwendung wieder. Hierbei beginnt die Benutzerführung im *EntryPoint* und kann über Aktionen auf der Benutzeroberfläche entlang der Pfeile erfolgen.

Zudem übernehmen die Controller das beschriebene Binding von Attributen und Elementen der Benutzeroberfläche. Über eine Listener-Struktur reagieren die Controller auf Eingaben mit den passenden Operationen auf der Datenhaltungsschicht.

Neben den Controllern sind in der Kontrollschicht noch die Klassen *PreferenceManager* und *RoundTimer* vorhanden, die sich um die Einstellungen des aktuellen Benutzers und die Rundenzeiten kümmern.

3.4 Verbindung der Komponenten

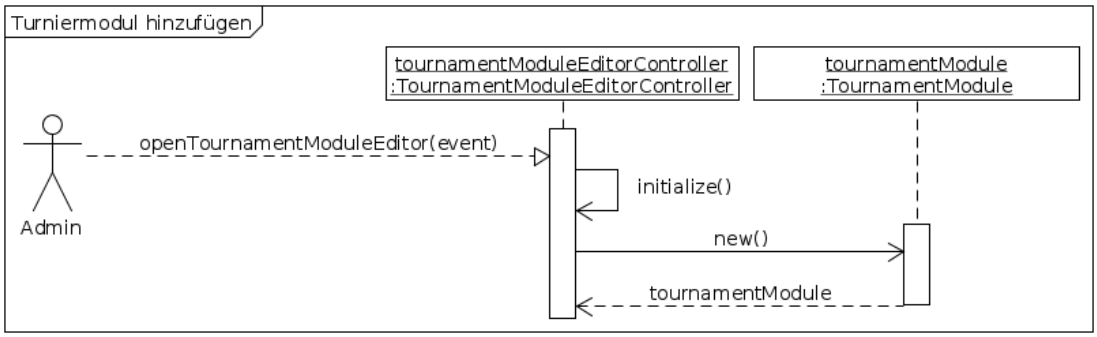
Die beiden Schichten View und Controller sind über eine Listener-Struktur miteinander verbunden, wobei Eingaben des Nutzers einen Callback zur Folge haben.

Die Kontrollstruktur hält für den Zugriff relevante Instanzen aus der Datenhaltungsschicht bereit.

Außerdem sind bestimmte Attribute untereinander und Elemente der Benutzeroberfläche über *JavaFX-Bindings* miteinander verknüpft. Hierzu werden als Attribute an den relevanten Stellen *JavaFX-Properties* verwendet.

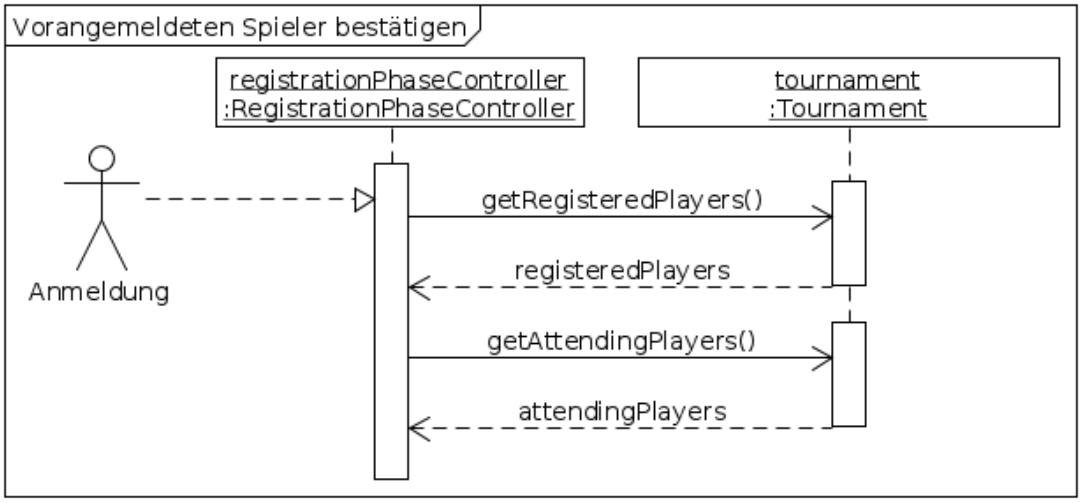
3.5 Sequenzdiagramme

3.5.1 Turniermodul hinzufügen



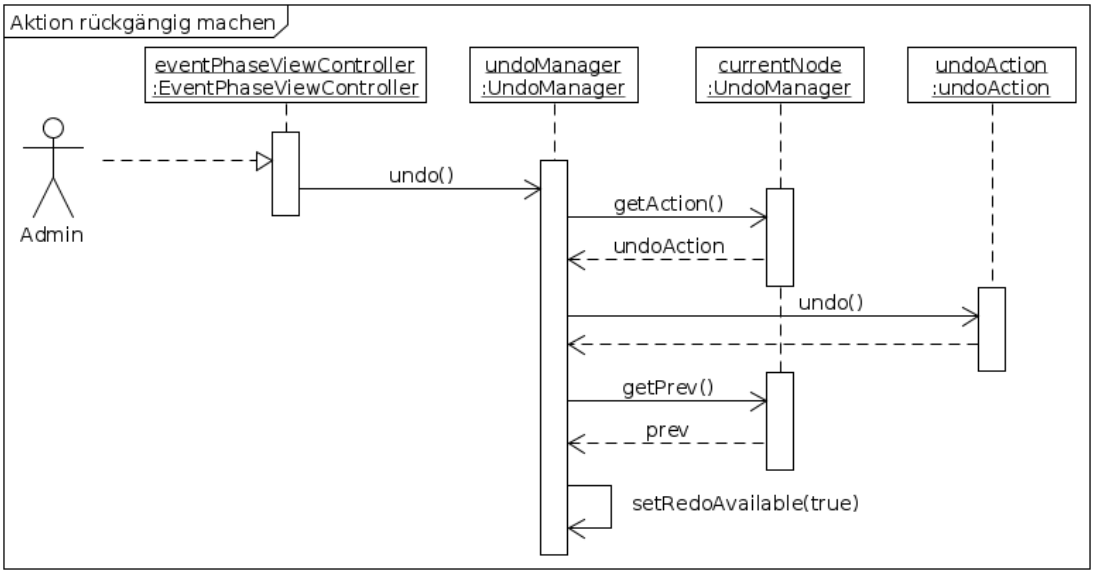
Sendet der Administrator über die Oberfläche den Befehl an den *TournamentModuleEditorController*, ein neues Regelmodul zu erstellen, so wird dieser zunächst initialisiert. Daraufhin wird eine neue Instanz der Klasse *TournamentModule* erstellt und an den Controller zurückgegeben. Dieser stellt nun die Oberfläche dar, in der der Administrator das Modul editieren kann.

3.5.2 Vorangemeldeten Spieler bestätigen



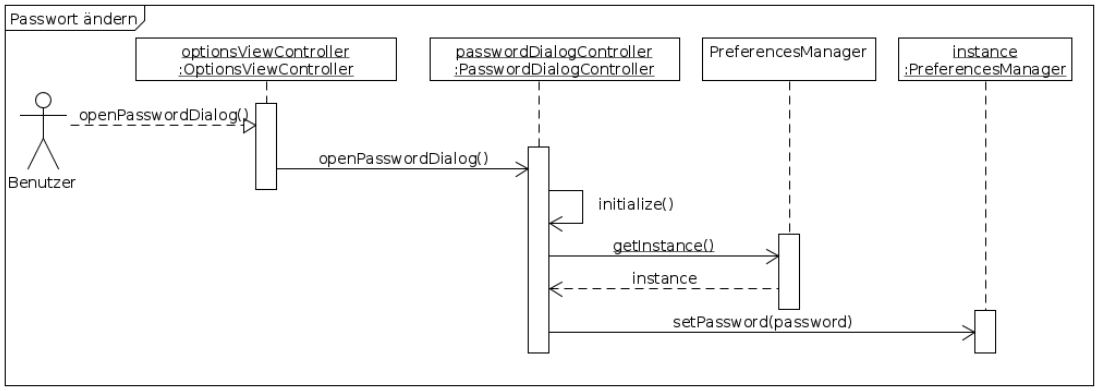
Um einen vorangemeldeten Spieler durch eine Anmeldung über den *RegistrationPhaseController* zu bestätigen holt der Controller die Liste aller registrierten und teilnehmenden Spieler vom Turnier-Objekt und kopiert den anzumeldenden Spieler in die Liste der teilnehmenden Spieler.

3.5.3 Aktion rückgängig machen



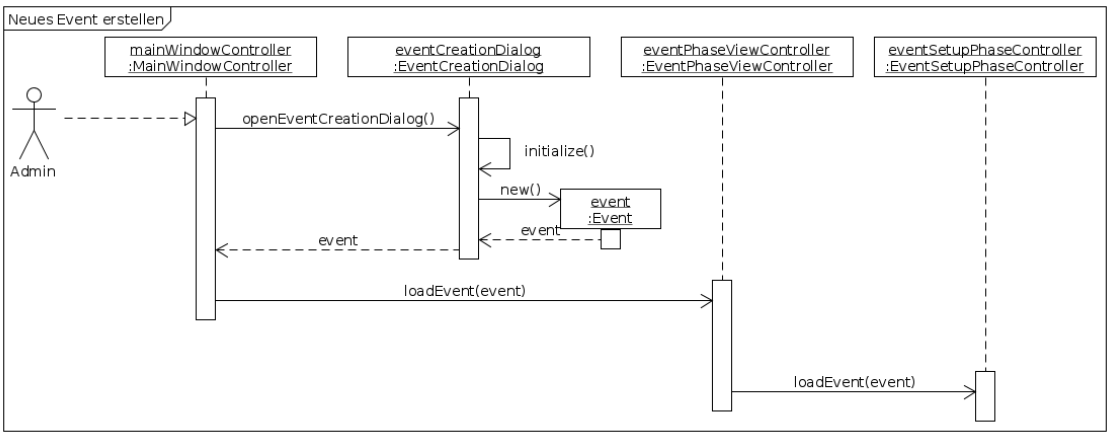
Will ein Nutzer eine Aktion im *EventPhaseViewController* rückgängig machen, so wird der zugehörige Befehl an die Instanz des *UndoManagers* gesendet. Dieser holt sich die aktuelle Aktion auf der Benutzeroberfläche von der Liste von Aktionen und macht diese rückgängig. Zuletzt wird noch das Wiederherstellen dieser Aktion möglich gemacht.

3.5.4 Passwort ändern



Das Ändern des Passworts findet über einen Aufruf des *OptionsViewControllers* statt. Dieser öffnet wiederum einen *PasswordDialogController*, der initialisiert wird und sich dann die Singleton-Instanz vom *PreferencesManager* holt. Über diese wird schließlich das neue Passwort gesetzt. Der *PreferencesManager* übernimmt hierbei Operationen wie das Hashing des Passworts.

3.5.5 Neues Event erstellen



Will der Administrator ein neues Event über den *MainWindowController* erstellen, so wird zunächst ein *EventCreationDialog* geöffnet und initialisiert. Dieser erstellt ein neues *Event*, das der *MainWindowController* dann an einen *EventPhaseViewController* weitergibt. Das neue Event wird schließlich in einem *EventSetupPhaseController* dargestellt.

3.6 Zuständigkeiten

Die Aufgabe der Führung der Implementierung der Komponenten wird folgendermaßen aufgeteilt:

- *Model*: Fabian Biester
- *View*: Jonas Auer
- *Controller*: Jan Tagscherer

4 Externe Schnittstellen

4.1 Dauerhafte Datenspeicherung

Die Speicherung von Daten findet über das *XML*-Dateiformat statt und wird von der Klasse *FileManager* übernommen. Diese bietet statische Methoden, die relevante Objektgeflechte speichern und aus validen Dateien laden kann.

4.1.1 Speicherung eines Events

Es werden die folgenden Dateistrukturierungen verwendet und in *Zip*-Dateien mit der Dateiendung *.tmf* zusammengefasst. Ein solches Dateipaket beschreibt ein komplettes Event mit allen seinen Turnieren und Teilnehmern.

Je nachdem ob das komplette Event auf einem Administrationsrechner gespeichert werden soll oder ob Spielerdaten an einen weiteren Anmeldungsplatz oder Turnierdaten an den Arbeitsplatz eines Turnierordners weitergegeben oder dort gespeichert werden soll werden nur die jeweils relevanten Auszüge aus den Dateien genutzt. Zudem lässt die Software über die Benutzeroberfläche keine Änderungen an Daten zu, die ein Nicht-Administrator nicht bearbeiten darf, da diese beim Zusammenfügen am Administrationsrechner schließlich nicht beachtet werden.

4.1.1.1 Event

Die Datei *Event.xml* speichert allgemeine Daten zum ganzen Event. Sie ist folgendermaßen strukturiert:

- metadata
  - name
  - location
  - date
    - start-date
    - end-date
  - event-administrators
    - administrator 1
      - name
        - first-name
        - last-name
      - mail-address
      - phone-number
    - ⋮
- tournaments
  - tournament 1
    - id
  - ⋮

4.1.1.2 Spieler

Die Datei *Players.xml* speichert alle Spieler, die zu einem jeweiligen Event vorangemeldet oder angemeldet sind und ist folgendermaßen strukturiert:

- player 1
  - id
  - name
    - first-name
    - last-name
  - mail-address
  - nick-name
  - starting-number
  - payed
  - disqualified
- ⋮

4.1.1.3 Turnier

Die Dateien *TournamentID.xml* speichern jeweils ein Turnier in folgender Struktur, wobei im Dateiname die eindeutige ID des Turniers angegeben ist.

- metadata
  - name
- tournament-administrators
  - administrator 1
    - name
      - first-name
      - last-name
    - mail-address
    - phone-number
    - ⋮
- registered-players
  - player-id 1
  - ⋮
- attendant-players
  - player-id 1
  - ⋮
- tournament-rounds
  - tournament-round 1
    - pairings
      - pairing 1
        - participants
          - player 1
            - id
            - scores
              - score 1
              - ⋮
            - ⋮
        - ⋮
    - ⋮
- tournament-rules
  - tournament-phases
    - phase 1
      - phase-number
      - number-of-rounds
      - pairing-system
      - cutoff-player-number-after-phase
      - ⋮

4.1.2 Speicherung von Regelmodulen

Neben den Events mit Turnieren und Spielern müssen auch noch die Vorlagen für Turniermodule gespeichert werden.

4.1.3 Regelmodul

Die folgendermaßen strukturierte Datei *RuleTemplateID.xml* speichert jeweils eine Vorlage für Turnierregeln, die zur Erstellung eines neuen Turniers verwendet werden können. Auch sie trägt eine eindeutige ID im Dateinamen.

- meta-data
  - name
  - description
- possible-scores
  - score 1
    - name
    - points
    - ⋮
- tournament-phases
  - phase 1
    - phase-number
    - number-of-rounds
    - pairing-system
    - cutoff-number-after-phase
    - ⋮

4.2 Externer Zugriff

Auf dem Gebiet des Turniermanagements wie es von **Tourney** durchgeführt wird sind keine zu bedenkenden Systeme vorhanden. Alle Eingaben finden über die Benutzeroberfläche statt und Ergebnisse werden nur direkt an den Nutzer ausgegeben. Daher erfolgt weder auf externe Daten noch auf **Tourney** von externer Seite ein Zugriff und derartige Schnittstellen müssen nicht eingeplant werden.

5 Versionshistorie

- Version 1.0 (06.12.2014)
  - Grundstruktur des Entwurfs erstellt
- Version 1.1 (07.12.2014)
  - Kapitel 1 hinzugefügt
- Version 1.2 (10.12.2014)
  - Kapitel 2 und 4 hinzugefügt
- Version 1.3 (11.12.2014)
  - Kapitel 3 hinzugefügt