PSE: Blockchain-basiertes E-Voting

Tim Fröhlich, Achim Kriso, Philipp Schaback, David Schuldes, Artem Vasilev
Phasenverantwortlicher: David Schuldes

26. Juni 2018

Inhaltsverzeichnis

1 Einleitung **1**

2 Architektur **1**
2.1 Interfaces zwischen den Paketen 3
2.2 Verwendete Libraries 4
2.3 Internationalisierung 4
2.4 Typische Abläufe 5

3 Model **12**
3.1 Paket: StateManagement 12
3.1.1 Election 12
3.1.2 VotingSystem 13
3.2 Paket: SDKConnection 13
3.2.1 Übersicht 13
3.2.2 Wichtige Elemente der SDKConnection 13
3.2.3 Typische Abläufe aus Sicht von SDKConnection 17

4 View **23**
4.1 Components 23
4.1.1 Diagramme 23
4.1.2 Tabellen 24
4.2 Supervisor View 26
4.2.1 Main View 26
4.2.2 Konfigurations View 27
4.3 VoterView 28

5 Control **29**

6 Exceptions **30**

7 Einteilung zur Implementierung **30**
7.1 Abhängigkeiten 30
7.2 Gantt-Diagramm 30

8 Änderung zum Pflichtenheft **31**
8.1 K2: Geheime Wahlen 31

1 Einleitung

Dieses Dokument beschreibt die Ergebnisse der Entwurfsphase im Rahmen des Moduls Praxis der Softwareentwicklung (PSE) am Karlsruher Institut für Technologie, das am Lehrstuhl für Anwendungsorientierte formale Verifikation von Professor Beckert ausgeschrieben wurde. Entworfen wurde die im Pflichtenheft definierte Software "Blockchain-basiertes E-voting".

Ziel der Software ist es, die Manipulation von Wahlergebnissen zu verhindern und den Wählern zu gewährleisten, dass ihre Stimme unverändert in die Wahl eingegangen ist. Ziel dieses Dokumentes ist es, zu erläutern wie die im Pflichtenheft spezifizierten Anforderungen softwaretechnisch umgesetzt werden sollen.

2 Architektur

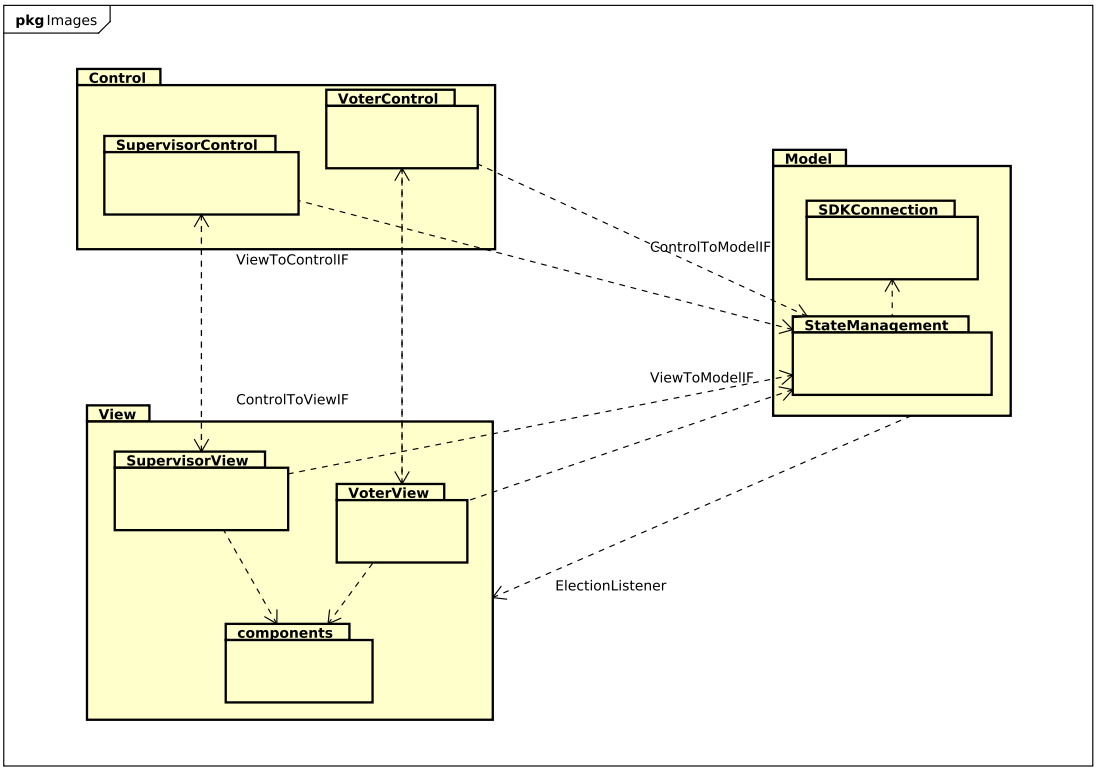

Abbildung 1: Übersicht der Pakete und deren Interaktion

Basierend auf der Model-View-Control Architektur ist die Software in ein Model-Paket,

View-Paket und Control-Paket unterteilt und jede Assoziation zwischen den Paketen durch eine Schnittstelle klar definiert.
Die Entwurfs-Architektur hat zwei Klienten, einen Wahlleiter- und einen Wähler-Klient. Die Schnittstellen, die von beiden Klienten nach außen angeboten werden, bestehen aus einer Hierarchie von drei Interfaces. Ein Interface welches die Gemeinsamkeiten der beiden Klienten erfasst und jeweils ein Interface für jeden Klienten der das allgemeine Interface um die benötigte Funktionalität erweitert. Dabei spiegelt der Name eines Interfaces eine unidirektionale Assoziation zwischen den Paketen die es repräsentiert wider. Beispielsweise ist *ControlToViewIF* das Interface, welches das Control-Paket benutzt um auf Funktionalität des View-Pakets zuzugreifen.
Das *ElectionDataIF* dient als Data-Transfer-Object. Es bietet alle getter-Methoden, um auf die Daten einer Wahl zuzugreifen. Dieses Interface wird von den Paketen Model und View implementiert, die von Rohdaten (beispielsweise aus der Blockchain), eine Wahl laden. Dadurch kann das Data-Transfer-Objekt an andere Pakete übertragen werden, ohne die Modularität des Entwurfs zu verletzen.
Das View- und Control-Paket stehen in einer bidirektionalen Assoziation zueinander. Beide Pakete haben außerdem eine Unidirektionale Assoziation zu dem Model-Paket. Die einzige Ausnahme zu diesem System an Schnittstellen ist der Callback-Listener *ElectionStatusListener*, der von dem Model benutzt wird um das View-Paket über den Zustand der Wahl zu benachrichtigen und von dem Blockchain-Netzwerk ausgeht.

View: Dieses Paket dient zur Darstellung der Benutzeroberfläche und benutzt das Java-Swing GUI-Framework für die Darstellung der einzelnen GUI Elemente. Außerdem enthält das Paket eine Reihe an eigenen Elementen, die für speziellere Anforderungen dieses Projektes konstruiert wurden. Unterschiede zwischen den beiden Klienten finden sich in der Darstellung der Benutzeroberflächen und der zugehörigen Funktionalität, deswegen sind sie in zwei Unterpakete unterteilt.

Control: Das Control-Paket nutzt die vom Java-Swing Framework bereitgestellten Listener, die vom Control-Paket an das View-Paket weitergeleitet werden, um direkt auf Benutzereingaben einzugehen. Sobald Control eine Eingabe empfängt werden Aufgaben abhängig davon, welcher Listener ausgelöst wurde über die Schnittstellen an das Model- oder View-Paket weitergeleitet. Wie das View Paket ist auch das Control-Paket in ein Subpaket für den Wahlleiter und ein Subpaket für den Wähler unterteilt. Die Subpakete enthalten jeweils eine spezielle Implementierung des *ViewToControl*-Interfaces und alle benötigten Listener.

Model: Das Model-Paket hat zwei Hauptaufgaben. Es verwaltet alle zustandsrelevanten Daten und hält die Schnittstelle zu dem Blockchain-Netzwerk über die vom Hyperledger-Framework gebotenen Schnittstellen. Diese zwei Aufgaben sind in zwei Subpakete unterteilt. Das SDKConnection Paket, welches die Schnittstellen des Hyperledger Frameworks auf die Anforderungen der Software anpasst und das StateManagement Paket, welches alle Daten bereitstellt und deren interne Logik enthält.

2.1 Interfaces zwischen den Paketen

- **ViewToModelIF** Das ViewToModelIF bietet der View ein Interface, um auf die Datenhaltung zuzugreifen. Es werden Methoden bereitgestellt, die es ermöglichen allgemeine Daten zur Wahl abzufragen. Die hiervon ausgehenden konkreten Interfaces SupervisorViewToModelIF und VoterViewToModelIF stellen weitere Methoden zur Verfügung. Diese fragen Daten ab, die ausschließlich für den Wahlleiter oder respektive nur für den Wähler relevant sind. Die Methode setElectionEndListener dient dem Festlegen eines ActionListeners für das Wahlende beim initialen starten des Programms.

- **ControlToModelIF** Das ControlToModelIF bietet der Control ein Interface, um auf die Datenhaltung zuzugreifen. Es werden Methoden bereitgestellt, die es ermöglichen die Daten auf der Datenhaltung zu manipulieren und entsprechende Zugriffe auf das Blockchain Netzwerk auszuführen. Die hiervon ausgehenden konkreten Interfaces VoterControlToModelIF und SupervisorControlToModelIF stellen zusätzlich Methoden zur Verfügung. Diese ermöglichen Netzwerkzugriffe und Datenmanipulation, die lediglich für den Wahlleiter oder respektive nur für den Wähler relevant oder erlaubt sind. Diese Interfaces erfüllen mitunter **F1, F2, F3, F4, F5, F6, F7, F8, F9, F10** aus Sicht des View-Pakets.

- **ControlToViewIF** Das ControlToViewIF bietet der Control ein Interface, um auf die GUI zuzugreifen. Die hiervon ausgehenden konkreten Interfaces VoterControlToViewIF und SupervisorControlToViewIF stellen zusätzlich Methoden zur Verfügung, die nur für den Wahlleiter oder respektive nur für den Wähler relevant sind. Diese Gruppe von Interfaces erfüllt zwei Funktionen: Manipulation der GUI bei relevanten Änderungen im System und das Abfragen von ausgefüllten Daten an der GUI. Diese Interfaces ermöglichen zudem das Übermitteln von Fehler- oder Erfolgsmeldungen an die GUI. Diese Interfaces erfüllen mitunter **F1, F11, F13** aus Perspektive des View-Pakets.

- **ViewToControlIF** Das ViewToControlIF und seine Subklassen dienen dem Weiterleiten aller notwendigen ActionListener an die GUI, damit diese dort an entsprechenden Komponenten registriert werden können.

- **ElectionStatusListener** Durch das ElectionStatus-Interface kann das Model-Paket der GUI den Status der Wahl mitteilen. So kann ein Update der Daten in der GUI angefragt werden oder die Beendigung der Wahl mittgeteilt werden. Dieser Listener sorgt mitunter für die Erfüllung von **F14, F15**.

- **ElectionDataIF** Das *ElectionDataIF*-Interface dient als Data-Transfer-Object zwischen den Model-, View- und Control-Paketen, um allgemeine Daten über die Wahl zu kommunizieren. Diese allgemeinen Daten enthalten: Den Namen, die Beschrei-

bung, den Zeitraum, die Kandidaten und die Beschreibungen der Kandidaten der Wahl. Da diese Daten genau einmal gesetzt werden sind keine Setter benötigt.

- **ConfigIssues** Das *ConfigIssues*-Interface dient ebenfalls als Data-Transfer-Object. Es enthält vier Methoden *getNameIssue(), getCandidateIssue(), getVoterIssue(), getTimespanIssue()*, die jeweils das Resultat der Prüfung der Benutzereingaben des Wahlleiters beim Erstellen einer Wahl enthalten. Die Prüfungs-Resultate werden in Form eines internationalisierten (dieser Begriff ist in Abschnitt 2.3 gesondert erläutert) Strings zurückgegeben, dieser wird dann im Fall einer ungültigen Eingabe als Fehlermeldung in der Konfigurations-View angezeigt.

2.2 Verwendete Libraries

Für das Projekt werden diverse Libraries benötigt und verwendet:

- javax.swing
- javax.json
- fabric-ca-java-sdk
- fabric-java-sdk

Im Go-Chaincode wird zusätzlich Hyperledger Fabrics *shim* benötigt:

- github.com/hyperledger/fabric/core/chaincode/shim

2.3 Internationalisierung

Internationalisierung erlaubt es, ohne großen Aufwand unterschiedliche Sprachen in dem Softwareprojekt zu unterstützen. Alle Texte werden automatisch in der Sprache des Benutzers angezeigt, außerdem wird der Objekt-Orientierte Entwurf durch zusätzliche Modularität verstärkt. Die Implementation geschieht mithilfe der von Java gebotenen *RessourceBundle*-Klasse. Ein allgemeiner String wird in einen internationalisierten String umgewandelt, der den Sinn des allgemeinen String in der entsprechenden Sprache formuliert. Dabei existiert für jede unterstützte Sprache eine Datei, die eine Key-Value-Map vom allgemeinen String zum internationalisierten String enthält. Diese Dateien werden automatisch von der *RessourceBundle*-Klasse geladen, um während der Laufzeit automatisch den richtigen String zu benutzen.

2.4 Typische Abläufe

- **Aufbau der Topologie, Abb. 2** Aufbau der Topologie bezieht sich in diesem Kontext auf die Initialisierung der für den Programmstart relevanten Modulen und deren Assoziationen. Beim initialen Starten der Anwendung (hier aus Sicht des Wahlleiters) wird zuerst die *SupervisorGUI*-Klasse erstellt. Daraufhin wird der Listener für den Status der Wahl erstellt. Danach wird das Model aufgebaut und der erstellte Listener registriert. Danach wird die Control erstellt und das Model dort registriert. Im Anschluss wird der Supervisor-Adapter erstellt und die Control an diesem registriert. Letztlich wird der anzuzeigende Panel erstellt und somit die Benutzeroberfläche angezeigt. Der Aufbau der Topologie aus Sicht des Wählers geschieht analog. Die Initialisierung der SDK Schnittstelle und deren Topologie erfolgt an dieser Stelle noch nicht, das geschieht bei der ersten Authentifizierung des Wahlleiters.

- **Import einer Wahlkonfiguration, Abb. 3** Nach Auslösen des *ImportConfigListeners* wird in der View nach dem ausgewählten Pfad gefragt. Dieser Pfad wird an das Model gegeben, welches die Datei liest und dort die abgespeicherten Daten in ihren Attributen ablegt. Schließlich wird die View vom *ImportConfigListener* dazu aufgefordert, die Daten aus dem Model zu holen und in ihrer ConfigGUI anzuzeigen. Dabei holt die *SupervisorGUI* die Wahldaten in Form eines *ElectionDataIF* Data-Transfer-Object und die, in der Konfiguration definierten Wähler, ebenfalls als Data-Transfer-Object. Diese beiden Daten-Objekte werden einzeln an die *ConfigGUI* gereicht, welche die im *ElectionDataIF* enthaltenen Daten extrahiert und in den entsprechenden *SupervisorGUIPanel* Objekten einsetzt.

- **Starten einer Wahl, Abb. 4** Sobald der Wahlleiter seine Konfiguration bestätigt hat, wird der entsprechende ActionListener ausgeführt. Dieser benachrichtigt das Model, die Wahldaten an SDKConnection zu übergeben, damit die Wahl starten kann. Ob dieser Prozess erfolgreich durchlaufen wurde ist an dem Rückgabe-Boolean erkennbar und der Wahlleiter wird dementsprechend benachrichtigt.

- **Authentifizierung, Abb. 5** Zuerst drückt der Wähler den Knopf "Weiter", daraufhin wird *actionPerformed()* aufgerufen. Durch *getAuthenticationPath()* wird der angegebene Pfad aus der GUI gelesen. Dann beginnt die Autorisierung durch den Aufruf von *authenticate()* auf einem *VoterElection*-Objekt. Daraufhin wird eine *VoterSDKInterfaceImpl*-Schnittstelle instanziiert und erhält zwei Parameter: Den Pfad zum Wähler-Zertifikat und einen Listener. Mit der *start()*-Methode wird ein neuer Thread der *SDKEventListenerImpl* erzeugt, welcher regelmäßig den Status der Wahl aus dem Netzwerk abfragt. Bei erfolgreich abgeschlossener Authentifizierung wird durch *showChoice()* auf der *VoterGUI* ein neues *VoterChoice* Panel erstellt und angezeigt. Es stellt die Benutzeroberfläche für den Wähler dar, über die er seine Stimmabgabe tätigen kann.

- **Wahl, Abb. 6** Bei betätigen des Wahl-Buttons durch den Wähler wird der ActionListener *VotedListener*, welcher im Control-Paket liegt, aufgerufen. Dieser holt sich daraufhin die Kandidaten-Auswahl des Wählers aus dem View-Paket mittels dem *ControlToViewIF*. Im Anschluss wird die Kandidaten-Auswahl an das Model weitergegeben, wo sie dann an das Blockchain-Netzwerk übermittelt wird. Bei erfolgreichem Speichern der Kandidaten-Auswahl im Blockchain-Netzwerk wird diese auch in der Datenhaltung so übernommen. Zuletzt führt der *VotedListener* bei erfolgreich abgegebener Stimme die *showWait()*-Methode auf der GUI aus, welche dafür sorgt, dass die Benutzeroberfläche, die für den Zeitraum zwischen Wahlabgabe und Wahlende bestimmt ist, für den Wähler angezeigt wird. Hierfür wird eine neue Instanz der *VoterWait*-Klasse erstellt, welche sich zuerst über den *VoterAdapter* die notwendigen Daten holt und daraufhin mithilfe des *VoterMajorityVotingSystemComponentManager* die notwendigen GUI-Komponenten zusammenstellt und anzeigt.

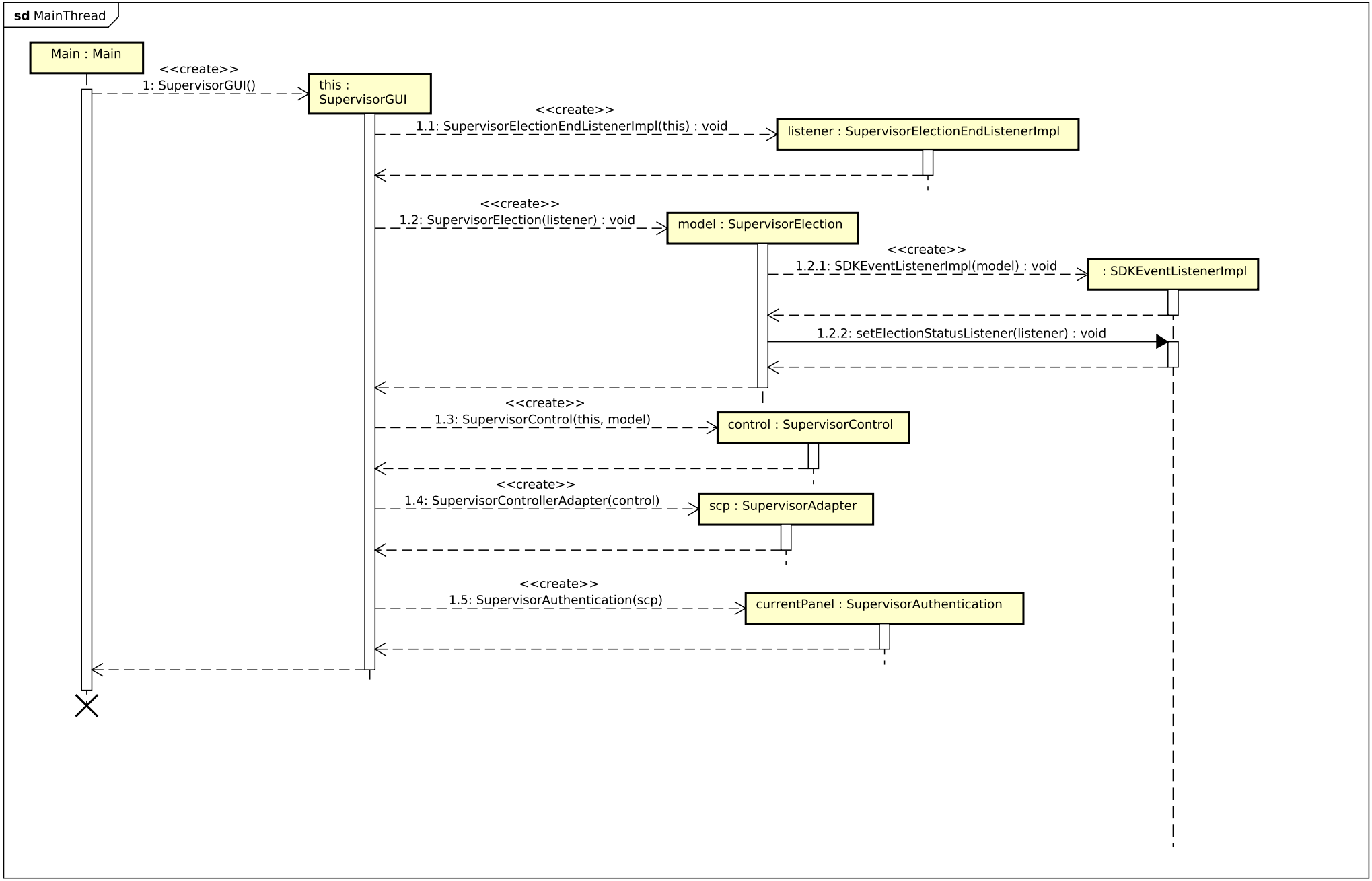

Abbildung 2: Start der Anwendung, Aufbau der Topologie

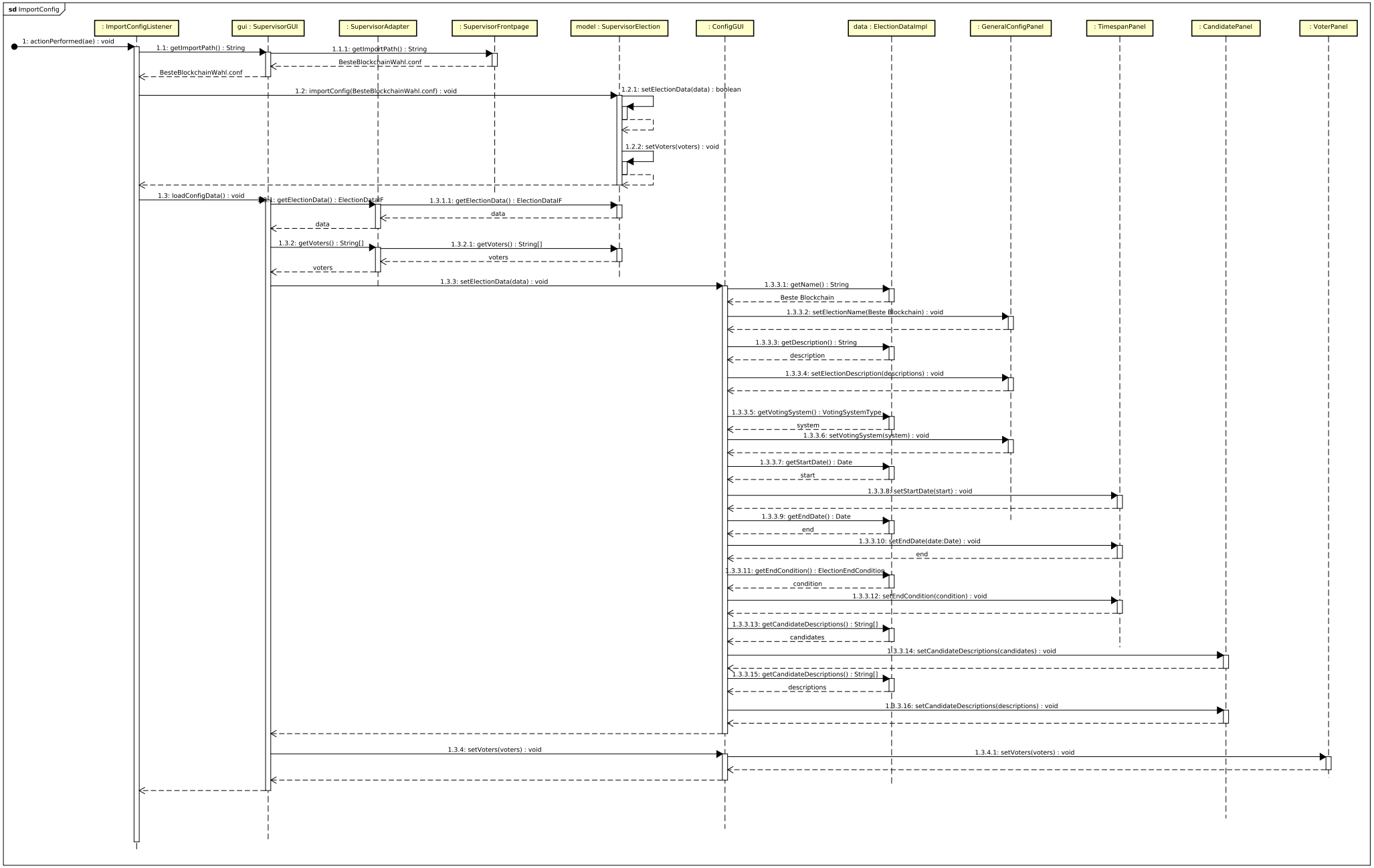

Abbildung 3: Importieren einer Wahlkonfiguration

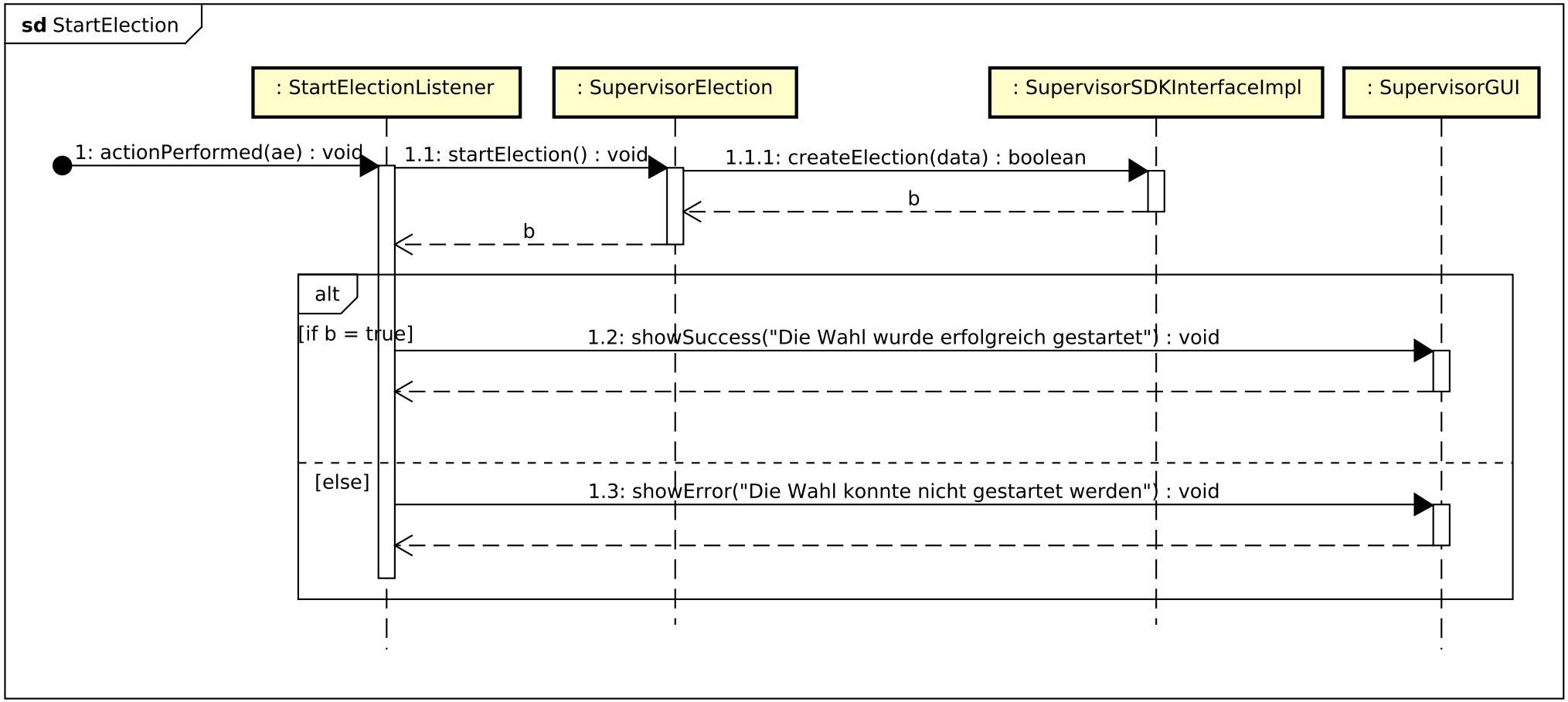

Abbildung 4: Starte einer Wahl

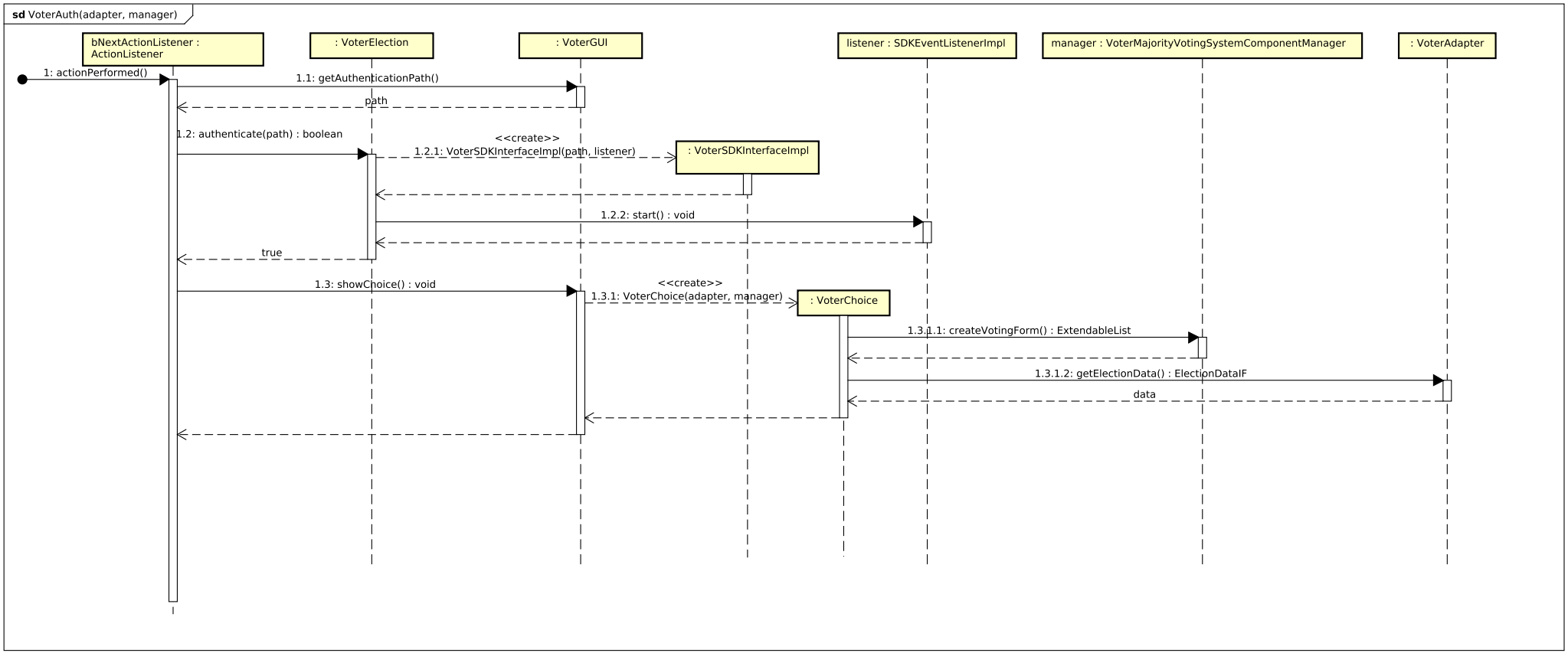

Abbildung 5: Authentifizierung eines Wählers gegenüber des Netzwerks

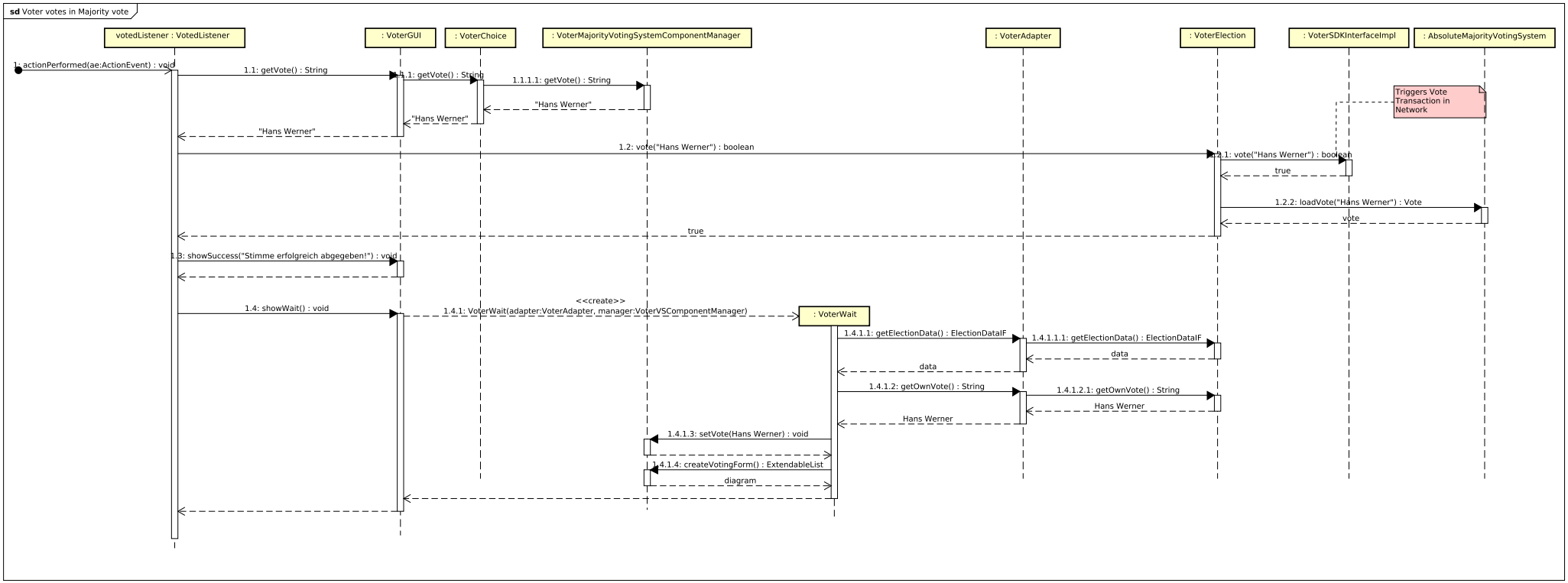

Abbildung 6: Abgabe einer Stimme durch einen Wähler

3 Model

Die zwei Subpakete in Model kommunizieren über die definierte Schnittstelle *SDKInterface* und einen Callback-Listener *SDKEventListener*. Um den Klienten darüber zu informieren, ob die Wahl aktiv ist, wird der *SDKEventListener* benutzt. Diesem wird ein Timer mit zufälliger Dauer zugewiesen. Wenn der Timer abgelaufen ist wird eine Anfrage an das Netzwerk gesendet, um zu ermitteln, ob die Wahl beendet ist. Auf diese Anfrage antwortet das Netzwerk mit einem Event an alle Klienten, welche daraufhin ihre GUI aktualisieren und ihren Timer zurücksetzen. Hierdurch wird die Netzwerklast gering gehalten, weil nur der Klient, dessen Timer die geringste Laufzeit hat, Netzwerkanfragen sendet, aber alle Klienten aktualisiert werden. Die Schnittstelle ist in drei Interfaces unterteilt. *SDKInterface* enthält die Gemeinsamkeiten der beiden Klienten und *SupervisorSDKInterface* und *VoterSDKInterface* enthalten jeweils die Unterschiede der Wahlleiter- und Wähler- Klienten.

3.1 Paket: StateManagement

Das StateManagement-Paket hat die Aufgabe alle wichtigen Zustandsdaten zu speichern, für die View zugänglich zu machen und die Logik der Daten zu implementieren. Dabei kommuniziert es mit dem SDKConnection-Paket um Daten von der Blockchain zu holen und Aktionen auf der Blockchain auszuführen.

3.1.1 Election

Im Zentrum dieses Paketes ist die *Election*-Klasse. Sie speichert die Kandidaten, Stimmen und allgemeine Wahldaten in Form des *ElectionDataIF*. Außerdem hält es den *SDKEventListener*, um über den Zustand der auf der Blockchain laufenden Wahl informiert zu werden.
Die *Election* Klasse wird erweitert durch:

- **SupervisorElection** Diese Klasse hält eine Assoziation zum *SupervisorSDKInterface* und jeweils eine Assoziation zu jedem wahlberechtigten Wähler, der durch eine *Voter*-Instanz modelliert wird.

- **VoterElection** Diese Klasse speichert die eigene Stimmabgabe eines Wählers, indem sie eine Assoziation zu einer *Vote*-Instanz hält. Außerdem hält *VoterElection* eine Assoziation zu *VoterSDKInterface*.

3.1.2 VotingSystem

Das Verhalten des Systems ist abhängig davon, welches Auszählverfahren für die Wahl festgelegt wird. Im StateManagement-Paket wird bestimmt welche Art der Stimmabgabe benutzt wird und wie der Gewinner ermittelt wird. Um diese Funktionen objekt-orientiert zu entwerfen existiert die abstrakte Klasse *VotingSystem*. Sie setzt voraus, dass alle Implementierungen eine *loadVote(vote : String)*-Methode und *determineWinner()*-Methode implementieren. Die *loadVote(vote : String)*-Methode dient dazu, Stimmen die in Form eines String vorliegen in Vote-Objekte umzuwandeln. *determineWinner()* evaluiert alle gespeicherten Stimmen in dem *Election*-Objekt und bestimmt, mithilfe des durch die Wahlkonfiguration festgelegten Auszählverfahrens, welcher Kandidat gewonnen hat. Damit wird **F15: Auszählung der Stimmen** erfüllt. Da das Wahlsystem beim Start eines Klienten nur als String vorliegt (entweder von der Blockchain geladen oder in der Konfigurationbenutzeroberläche eingegeben), wird die Klasse *VotingSystemBuilder* benutzt, um diesen String in ein *VotingSystem*-Objekt konvertieren. Zur Vereinfachung der Implementierung von *VotingSystem* haben *Vote*-Klassen eine *asString()*-Methode, welche ihren Zustand in einen String umwandeln. Solche Strings können durch die statische Methode *loadVote(vote : String)* wieder in ein äquivalentes Vote-Objekt umgewandelt werden. Der Aufbau eines solchen Strings ist abhängig vom gewählten VotingSystem. Hier wird eine Serialisierung des entsprechenden Vote-Objektes zu einem JSON-String vorgenommen, welcher vom VotingSystem selbst wieder deserialisiert werden kann. Die Übertragung der Stimmabgaben von View über Control zu Model erfolgt im gleichen JSON-String Format.

3.2 Paket: SDKConnection

3.2.1 Übersicht

Das Paket *SDKConnection* stellt die Verbindung der Software zum *Hyperledger Fabric SDK* und *Hyperledger Fabric CA SDK* dar. Diese SDKs übernehmen ihrerseits die Kommunikation mit dem Netzwerk. Hierdurch wird **F12** erfüllt. *SDKConnection* ist Teil des *Models*.

3.2.2 Wichtige Elemente der SDKConnection

SDKInterfaceImpl ist eine Implementierung der generische Schnittstelle *SDKInterface*. Sie wird erweitert durch die klientspezifischen Implementierungen *SupervisorSDKInterfaceImpl* und *VoterSDKInterfaceImpl*. Sie bietet Zugriff auf die von beiden Klienten gemeinsam genutzen Funktionen zur Popularisierung der Benutzeroberfläche mit Wahl-

informationen nach einem Neustart derselbigen und zum Absenden einer Anfrage des Wahlstatus.

SupervisorSDKInterfaceImpl ist eine Implementierung der Schnittstelle *SupervisorSDKInterface*. Sie bietet die zum Wahlleiter-Klienten spezifische Kommunikation mit dem SDK zum Erstellen und endgültigen Beenden der Wahl. Außerdem ermöglicht sie das Auslesen aller bisherigen Stimmen aus dem Ledger und das Erstellen neuer Benutzer.

VoterSDKInterfaceImpl ist eine Implementierung der Schnittstelle *VoterSDKInterface*. Sie bietet die zum Wähler-Klienten spezifische Kommunikation mit dem SDK zum Abgeben einer Stimme und Auslesen der eigenen Stimme aus dem Ledger.

AppUser repräsentiert eine Person aus der Sicht des SDKs. Die Klasse realisiert die Schnittstellen *User* des Hyperledger SDKs und *Serializable* des Java SDKs. *SupervisorSDKInterfaceImpl* erstellt Objekte dieses Typs und serialisiert sie zur Weitergabe an die Wähler.

Transactions beinhaltet die verschiedenen Transaktionen, die mittels des SDKs zum Netzwerk gesendet werden können. Die in den Schnittstellen gebotenen Methoden benutzen jeweils eine dieser Klassen, um die entsprechende Transaktion zu kapseln. Die abstrakte Klasse *Transaction* bietet die Schablonenmethode *getFunctionName*, deren Implementierungen den Funktionsnamen der gegenstelligen Funktion im Chaincode zurückgeben. Dieser ist Teil jeder Anfrage an die Peers. Die Transaktionen werden grundsätzlich in zwei Kategorien aufgeteilt:

1. **QueryTransaction**: Transaktionen, die nur Daten anfragen, aber keine Änderungen am Datenbestand des Ledgers vornehmen. *QueryTransaction* bietet die Schablonenmethode *parseResultString*, deren Implementierungen das Parsen der Rückgabe übernehmen.

    - **MultiStringTypeQuery**: Implementiert *parseResultString* für Anfragen, deren Rückgabe als mehrere Strings zu interpretieren ist.

        AllVotesQuery: Gibt alle Stimmen zur Auswertung zurück.

    - **SingleStringTypeQuery**: Implementiert *parseResultString* für Anfragen, deren Rückgabe als ein einzelner String zu interpretieren ist.

        OwnVoteQuery: Gibt die Stimme des *AppUsers* zurück.

- **ElectionDataTypeQuery**: Implementiert *parseResultString* für Anfragen, deren Rückgabe als *ElectionData* zu interpretieren ist.

  ElectionDataQuery: Gibt alle Metadaten der Wahl zurück.

2. **InvokeTransaction**: Transaktionen, die Änderungen am Datenbestand des Ledgers vornehmen, aber keine Daten zurückgeben. Die Konstruktoren dieser Klassen wandeln die übergebenen Parameter in ein JSON-String um, welcher dann durch *InvokeTransaction.invoke()* an das SDK übergeben wird.

   - **VoteInvocation**: Gibt die Stimme des *AppUsers* ab.
   - **InitializationInvocation**: Initialisiert die Wahl auf dem Ledger
   - **ElectionStatusUpdateInvocation**: Löst ein Update-Event im Chaincode aus, welches von allen *ElectionStatusListenern* empfangen wird.
   - **DestructionInvocation**: Startet das Netzwerk neu, um die Wahl endgültig zu beenden.

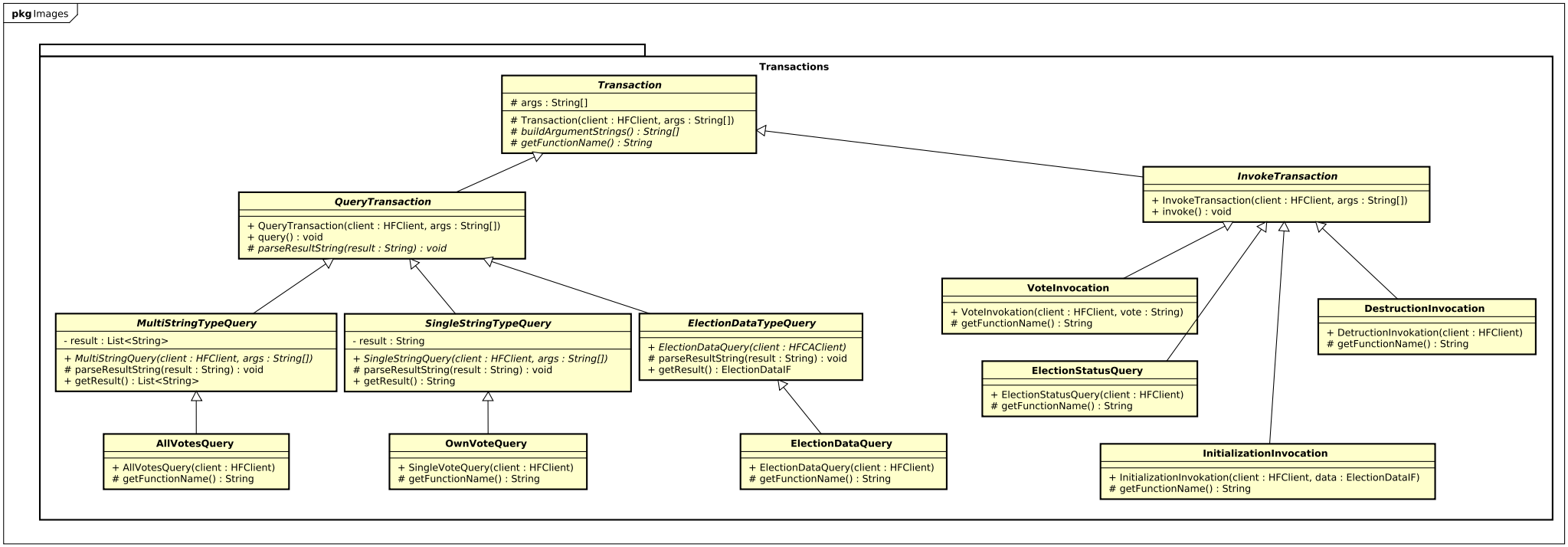

Abbildung 7: Das Transactions-Paket

3.2.3 Typische Abläufe aus Sicht von SDKConnection

Im Nachfolgenden sollen die Teile der im Pflichtenheft beschrieben Testfallszenarien beschrieben werden, welche SDKConnection betreffen. Sie erfüllen außerdem die funktionalen Anforderungen **F1, F2, F3, F4, F6, F8, F16** aus Sicht von *SDKConnection*.

Zuallererst registriert sich der Wahlleiter im Netzwerk (siehe Abb. 8). Dazu erstellt das StateManagement eine Instanz von *SupervisorSDKInterfaceImpl* und übermittelt dieser den Bootstrap-Username und das Bootstrap-Passwort. Diese wiederum erstellt eine Instanz von *HFCAClient*, der Hyperledger Fabric CA Schnittstelle, um die Identität des Wahlleiters (Admins, aus der Sicht des SDKs) zu registrieren. Mit dieser Identität (*Enrollment*) wird ein AppUser erstellt, welcher serialisiert und im mitgegebenen Dateipfad abgespeichert wird.

Als nächstes fügt das StateManagement die Wähler hinzu (siehe Abb. 9). Dazu ruft es die Methode *createUser* für jeden User auf, welche einen *RegistrationRequest* anlegt und mit diesem eine Identität beim *HFCAClient* registriert. Der damit gebaute *AppUser* wird serialisiert an den übergebenen Dateipfad geschrieben. Diese Dateien sind die Zertifikate, die an die Wähler verteilt werden müssen.

Dann werden die Wahldaten (*ElectionDataInterface*) festgelegt, hierzu dient die Methode *createElection* (siehe Abb. 10). Diese erstellt die entsprechende Transaktion aus dem Transaction-Paket, *InitializationInvocation*, welche wiederum die übermittelten Wahldaten in einen JSON-String verpackt. Mit der Methode *invoke* wird die Transaktion ans SDK und von diesem an das Netzwerk gesendet. Damit ist das Erstellen einer Wahl aus der Sicht von *SDKConnection* abgeschlossen.

Aus der Sicht des Wählers läuft die Registrierung im Netzwerk ähnlich: Das StateManagement erstellt eine Instanz von *VoterSDKInterfaceImpl* und übermittelt den Dateipfad zur vom Wahlleiter erhaltenen, serialisierten *AppUser*-Klasse. Als nächstes fordert das StateManagement die Daten der Wahl an. Dies geschieht über die Methode *getElectionData*.

Gibt der Wähler seine Stimme ab, so ruft das StateManagement *vote* auf (siehe Abb. 11). Analog zu Abb. 10 wird auch hier eine Instanz des richtigen Typs aus dem Transaction-Paket erstellt, von welchem auch hier die Parameter in einen JSON-String gewandelt und als *TransactionProposalRequest* an das SDK gesendet werden.

Zur Auswertung der Wahl ruft das StateManagement für den Wahlleiter die Methode *getAllVotes* auf (siehe Abb. 12). Diese erstellt die entsprechende Transaktion *AllVotesQuery*, welche mit der Methode *query* ans SDK und von diesem an das Netzwerk gesendet wird. Das Resultat der Anfrage wird als JSON-String empfangen, entpackt und an das StateManagement zurückgegeben.

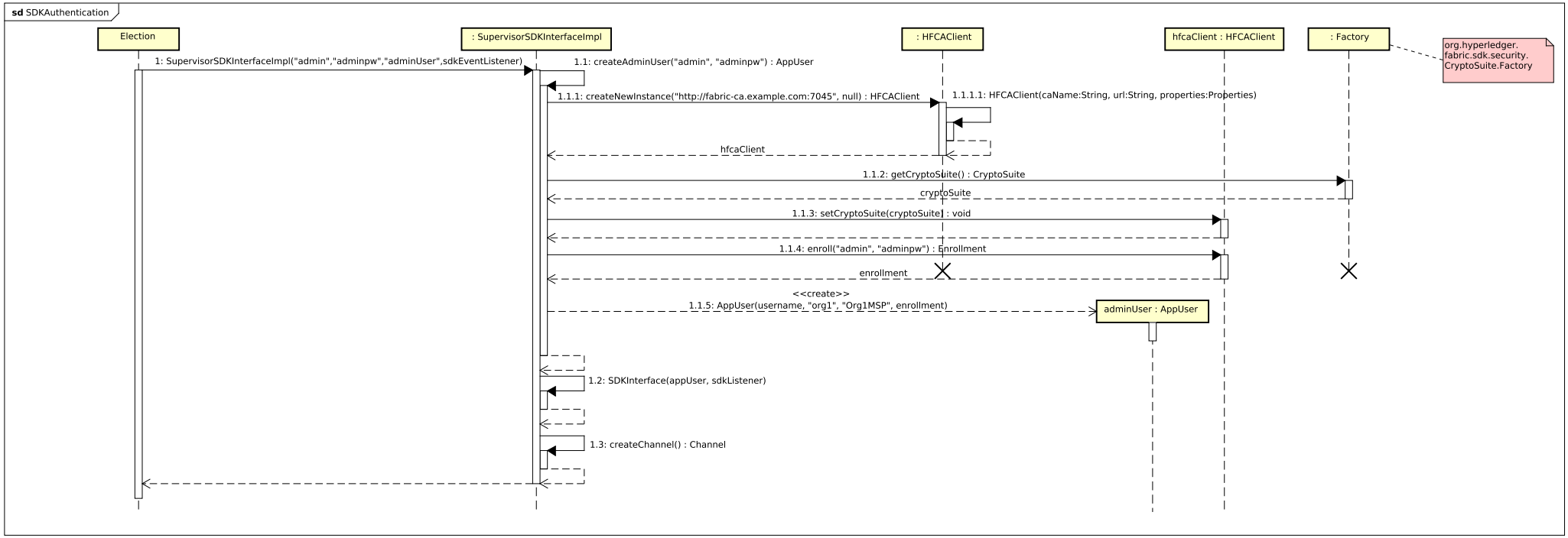

Abbildung 8: Erste Anmeldung des Wahlleiters im Netzwerk

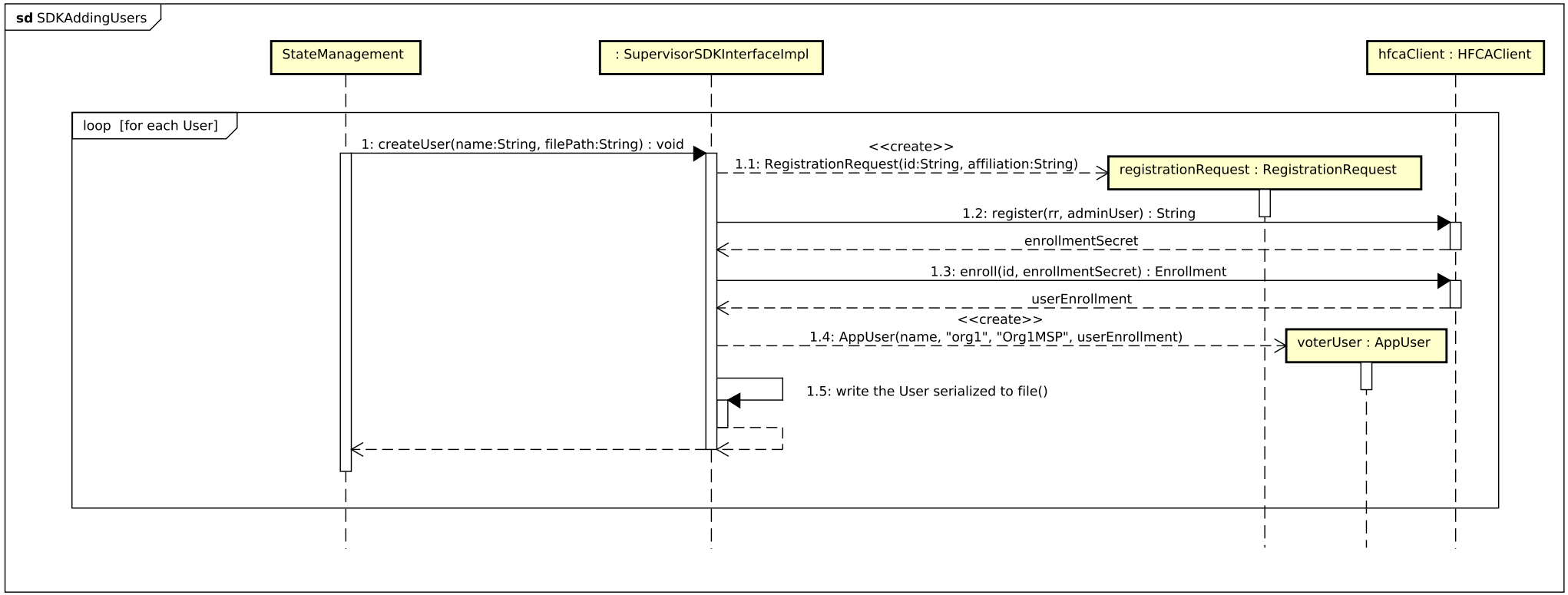

Abbildung 9: Hinzufügen von neuen Wählern durch den Wahlleiter

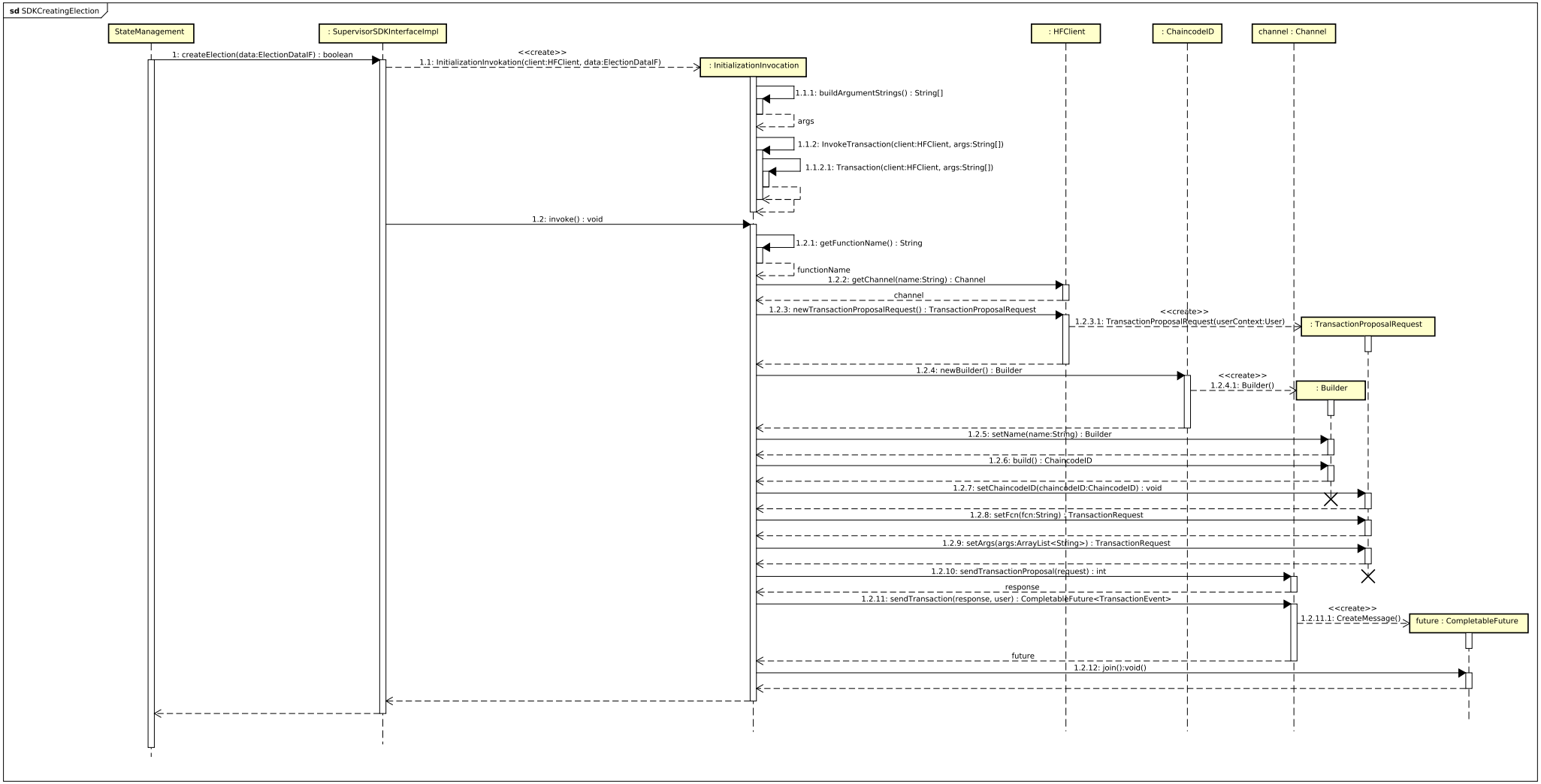

Abbildung 10: Erstellen einer Wahl durch den Wahlleiter

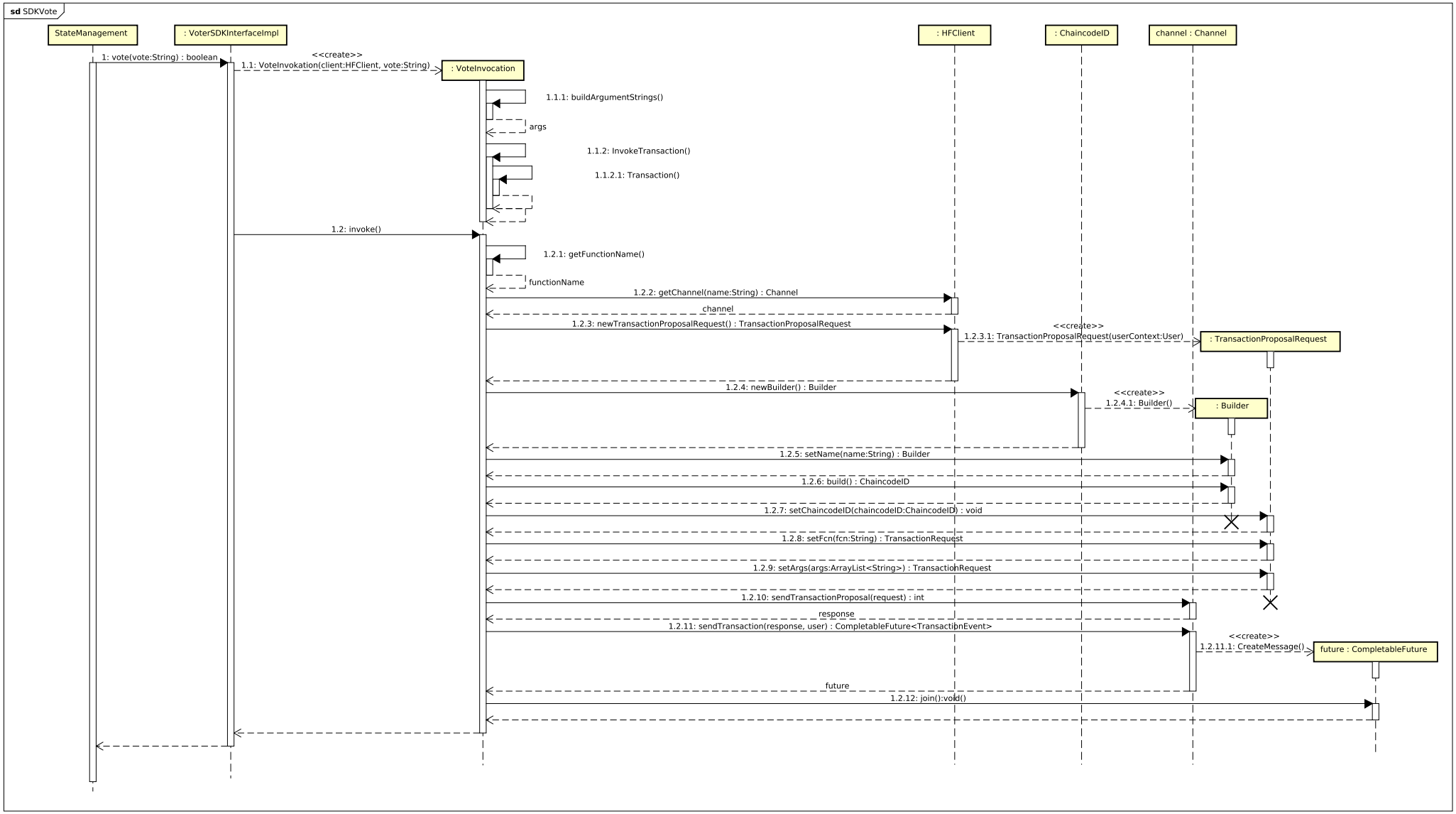

Abbildung 11: Abgabe einer Stimme durch den Wähler

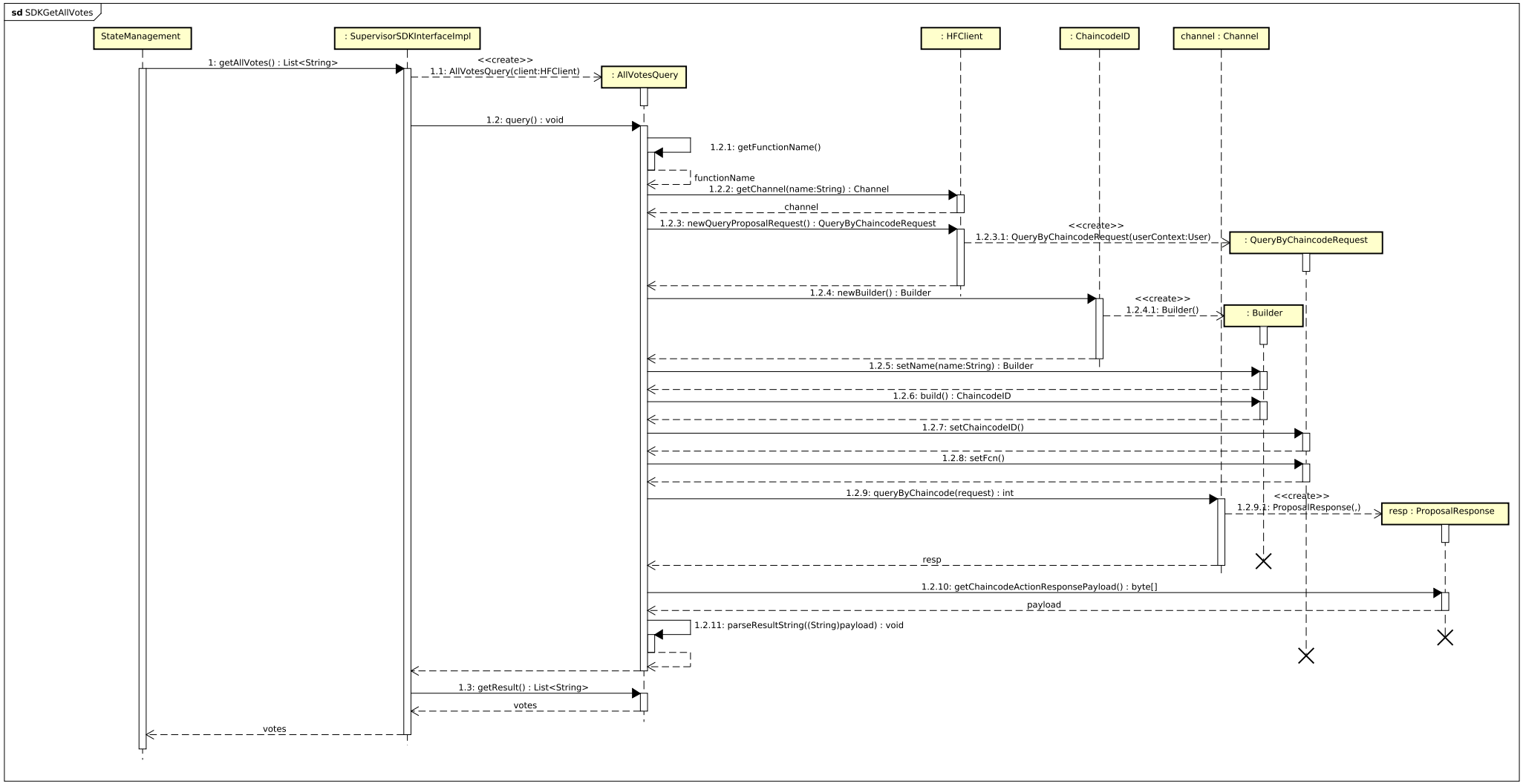

Abbildung 12: Auslesen aller Stimmen durch den Wahlleiter

4 View

4.1 Components

Dieses Unterpaket stellt dem Rest des View-Paketes Diagramme und ein modulares Tabellen-System bereit.

4.1.1 Diagramme

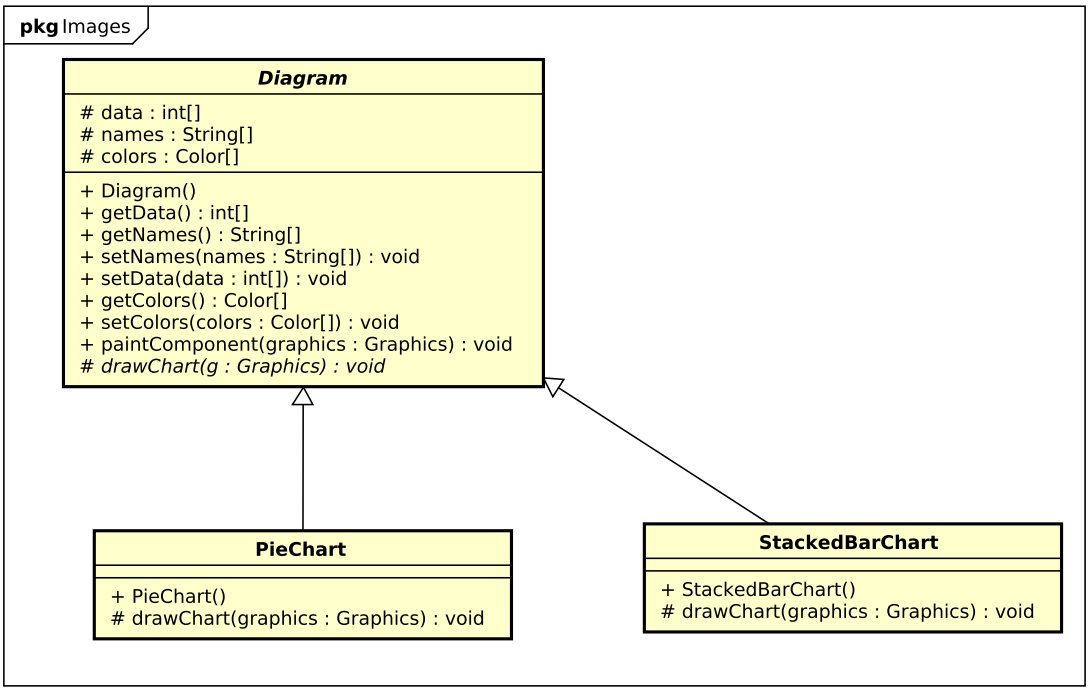

Abbildung 13: Klassendiagram der Diagramme

Diagramme werden vom View-Paket benötigt, um die Ergebnisse einer Wahl in einer benutzerfreundlichen Form darzustellen. Für die unterstützten Wahlsysteme sind unterschiedliche Formen der Visualisierung sinnvoll. Deswegen wird eine allgemeine Diagramm-Schnittstelle geboten. Damit kann das Diagramm die benötigten Daten entgegennehmen und durch die Java Swing Graphics Schnittstelle, die in der Schablonenmethode *drawChart(g : Graphics)* überreicht wird, zeichnen. Es ist jeder Implementation dieser Schnittstelle überlassen, wie diese Daten zu interpretieren sind. Um die im Pflichtenheft beschriebenen Wahlsysteme zu unterstützen, existieren die Implementationen *PieChart* für die Mehrheitswahlsysteme und *StackedBarChart* für Instant Runoff Voting.

4.1.2 Tabellen

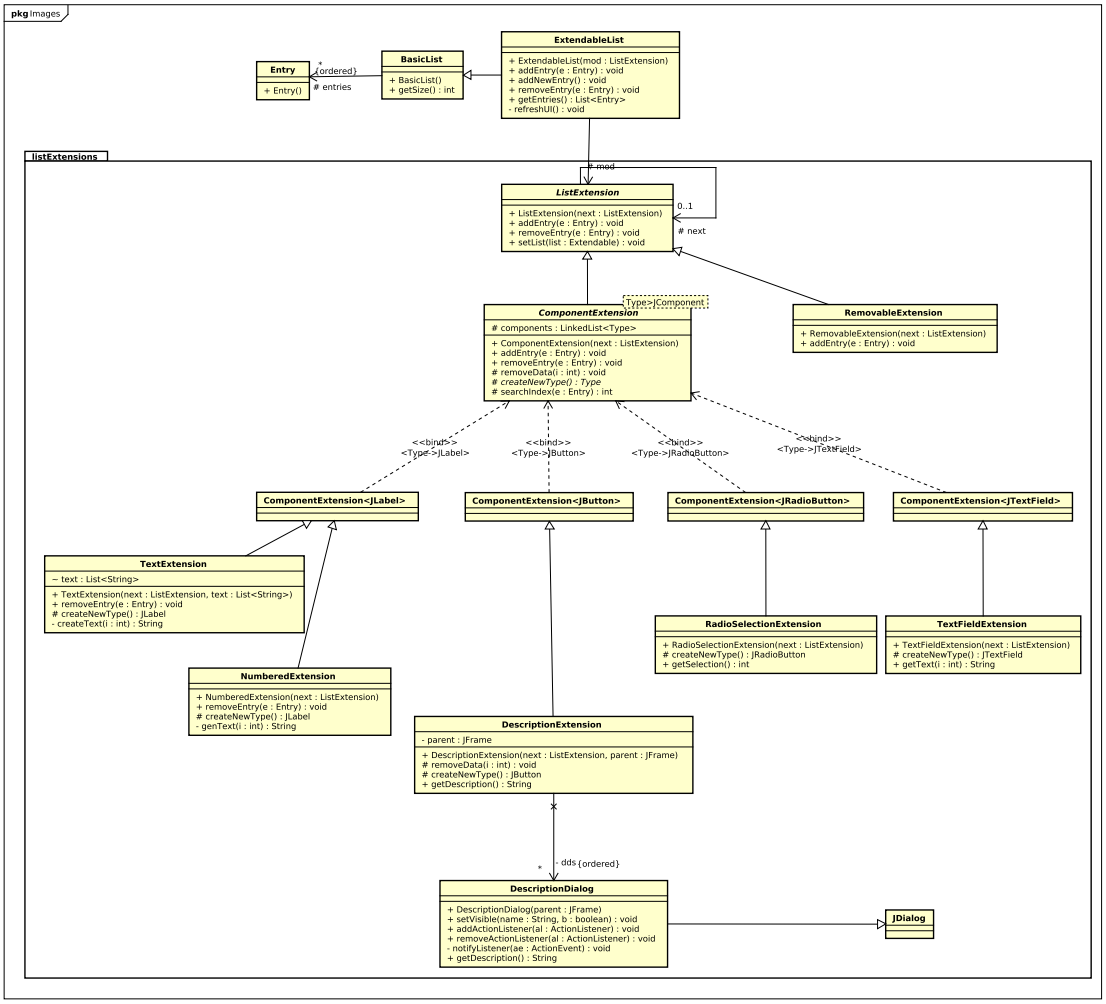

Abbildung 14: Klassendiagram des Tabellensystems

Tabellen mit unterschiedlicher Struktur und Funktionalität werden in verschieden Bereichen der View benötigt (beispielsweise Wahlkonfiguration, Stimmabgabe und Ergebniseinsicht). Um redundanten UI Quellcode zu vermeiden wurde eine erweiterbare Tabelle entworfen, welche aus Verschiedenen Modulen (Erweiterungen) besteht mit denen sich alle bestehenden Anforderungen dieses Projektes erfüllen lassen.
Grundlage ist eine Liste von Einträgen. Jeder Eintrag besteht aus den gleichen Komponenten. Ein Eintrag ist durch die Klasse *Entry* modelliert. Die Klasse *BasicList* implementiert das grundlegende Verhalten einer Liste. Dieses erfasst die Möglichkeit, durch die Tabelle zu scrollen, wenn sie nicht ganz in ihren Bereich passt und bietet die normale Schnittstelle einer jeden anderen Java Swing Komponente damit sie von dem Framework ohne Schwierigkeiten in die GUI integriert werden kann.

Die Modularität dieses Tabellen Systems wird durch die Klasse *ExtendableList* implementiert. Diese Klasse ist eine Erweiterung von *BasicList*. Objekte, die eine Instanz von *ListExtension* sind, können durch den Konstruktor von *ExtendableList* zu der Tabelle hinzugefügt werden. Erweiterungen sind mithilfe des Dekorierer-Entwurfsmusters umgesetzt. Sie werden miteinander verknüpft, indem jede Erweiterung eine andere durch ihren Konstruktor bekommen kann. Diese Kette an Erweiterungen wird schließlich dem Konstruktor der *ExtendableList* überreicht, der die von den Erweiterungen ausgehende Funktionalität umsetzen kann. Die Schnittstelle *ListExtension* benachrichtigt jede Erweiterung, wenn in der Tabelle ein Eintrag entfernt oder hinzugefügt wird. Wenn ein neuer Eintrag hinzugefügt (*addEntry(e : Entry)*) wird, wird ein leeres Objekt vom Typ *Entry* durch die Kette der Erweiterungen gereicht und jeder Erweiterung die Möglichkeit geboten, eigene Java Swing Komponenten auf diesem Eintrag zu platzieren. Sobald dieser Eintrag durch die komplette Kette gewandert ist, wird er in der Tabelle platziert. Es ist die Aufgabe jeder Erweiterung ihre Komponenten selbst zu verwalten, da die eigentliche Tabelle diese nicht kennt. Da viele Tabellen-Erweiterungen nur eine Komponente zu einem Eintrag hinzufügen ist diese spezielle Funktionalität in der Klasse *ComponentExtension* implementiert. Sie bietet außerdem eigene Schablonenmethoden über welche die Implementation starken Einfluss auf das konkrete Verhalten dieser Komponente nehmen können. Die Schablonenmethode *newType()* erlaubt der Implementation konkret zu bestimmen genaue welche Komponente zu dem nächsten Entry hinzugefügt werden soll. *removeData()* löscht die Komponente die sich auf dem Eintrag befindet welcher gerade gelöscht wird. Diese Methode erlaubt es einem Objekt von *ComponentExtension* seinen Zustand aufzuräumen.
Folgende Implementation der *ListExtension* werden benötigt:

- **TextExtension**: Fügt in jedem Eintrag ein *JLabel* ein, welches einen vordefinierten Text anzeigt. Dieser Text wird über ein Liste aus Strings in dem Konstruktor übergeben.

- **NumberedExtension**: Jeder Eintrag wird von oben nach unten nummeriert.

- **DescriptionExtension**: Eine Schaltfläche auf jedem Eintrag öffnet ein Dialog Fenster, in dem eine Beschreibung geschrieben werden kann.

- **RadioSelectionExtension**: Jeder Eintrag enthält einen Radiobutton und ist zusammen mit dem äquivalenten Radiobutton in den anderen Einträgen in einer *ButtonGroup*. So kann der Benutzer einen Eintrag auswählen.

- **TextFieldExtension**: Ein Textfeld wird auf jedem Eintrag angezeigt.

- **RemovableExtension**: Es ist möglich für den Benutzer Einträge zu löschen, indem er auf die von der Erweiterung hinzugefügten Schaltfläche drückt.

4.2 Supervisor View

Die Supervisor-View lässt sich unterscheiden in die Main-View und Konfigurations-View. Die Konfigurations-View ist ein Menü, in dem der Wahlleiter alle Einstellungen einer Wahl setzen kann. Die Main-View stellt die Benutzeroberfläche für das Authentifizieren, Wahlen importieren, die Ergebnis Einsicht und das endgültige Beenden von Wahlen dar. Sie verwaltet die Konfigurations-View und leitet Befehle von der Schnittstelle an die Konfigurations-View weiter.

Die Schnittstellen von Control und Model werden in der Adapter Klasse *SupervisorAdapter* vereinigt. Ein Objekt dieser Klasse wird an alle Supervisor-View-Klassen überreicht, die auf die Schnittstelle zugreifen müssen. Dies verringert die Anzahl an Objekten die durch alle Konstruktoren gereicht werden müssen und reduziert die Kohäsion zwischen den MVC-Paketen.

4.2.1 Main View

Der zentrale Baustein der Main-View ist die Klasse *SupervisorGUI*. Sie ist eine Erweiterung der Java-Swing-Klasse *JPanel* und ist die Implementierung der *SupervisorViewToControlIF*-Schnittstelle. Die *SupervisorGUI*-Klasse zeigt ein Objekt der Instanz *SupervisorGUIPanel* an und kann zwischen unterschiedlichen Implementierungen wechseln. Jede *SupervisorGUIPanel*-Implementierung steht für einen bestimmten Zustand, in dem sich die Main View befindet.

- **SupervisorAuthentication**: Fragt nach einem Zertifikat oder Namen und Passwort, mit dem sich der Wahlleiter authentifizieren kann.

- **SupervisorFrontpage**: Gibt dem Wahlleiter die Möglichkeit, eine neue Wahl zu konfigurieren, oder eine schon konfigurierte Wahl zu importieren. Dieser Zustand kann nur eintreten, wenn keine Wahl aktiv ist.

- **SupervisorResult**: Dieser Zustand tritt ein, wenn eine Wahl aktiv ist. Dem Wahlleiter werden die derzeitigen Ergebnisse der Wahl angezeigt.

Die Zustände *SupervisorFrontpage* und *SupervisorResult* werden durch *SupervisorControlToViewIF* über die Methoden *showFrontpage()* und *showResults()* gesetzt. Da das Authentifizieren nur einmal beim Programmstart passiert, kann der Zustand *SupervisorAuthentication* nicht von Control gesetzt werden.

Der *SupervisorResult*-Zustand zeigt eine Tabelle und ein Diagramm der Wahlergebnisse an. Da diese Tabelle und Diagramm abhängig vom festgelegten Wahlsystem sind,

gibt es die *SupervisorVSComponentManagerBuilder*-Klasse. Sie generiert, abhängig vom Wahlsystem, eine Instanz von *SupervisorVSComponentManager*. Implementationen dieser Klasse generieren das richtige Diagramm und die richtige Tabelle. Diese GUI Komponenten werden dann in das Layout des *SupervisorResult* integriert.

4.2.2 Konfigurations View

Die verwaltende Klasse der Konfigurations View ist *ConfigGUI*. Sie zeigt ein Menü an, welches durch verschiedene, vertikal angeordnete, Tabs navigiert werden kann (siehe Abb. 5-9 im Pflichtenheft). Dieses Tab-Layout und das damit verbundene Verhalten ist durch die Klasse *VerticalTabs* modelliert. Jeder Tab kann eine Instanz von *ConfigPanel* anzeigen. Diese Klasse enthält außerdem die 'Weiter' und 'Abbrechen' Schaltflächen. Die Schablonenmethoden *initComponents()* und *buildLayout()* stellen den Implementierungen dieser Klasse eine einfache Schnittstelle bereit, damit sie ihre Komponenten und deren Layout selbst zu bestimmen können. Folgende Implementation werden benötigt um die Funktionalen Anforderungen zu erfüllen:

- **GeneralConfigPanel**: Bietet Eingabeflächen um den Namen und Beschreibung festzulegen. Außerdem kann das Wahlsystem bestimmt werden, wodurch **F3: Auswahl des Auszählungsverfahren** aus Sicht der Benutzeroberfläche modelliert ist.

- **TimespanPanel**: In diesem Panel kann der Wahlleiter den Zeitraum festlegen, in dem die Wahl aktiv ist. *GeneralConfigPanel* und *TimespanPanel* erfüllen **F2: Allgemeine Konfiguration der Wahl**.

- **CandidatePanel**: Der Wahlleiter kann eine Liste an Kandidaten und deren Beschreibung festlegen. Hiermit ist **F6: Hinzufügen von Kandidaten** aus Sicht der Benutzeroberfläche erfüllt.

- **VoterPanel**; Der Wahlleiter kann eine Liste an erlaubten Wählern bestimmen. Dadurch ist **F4: Hinzufügen von Wählern** aus Sicht der Benutzeroberfläche erfüllt.

- **FinishPanel**: Dem Wahlleiter wird eine Übersicht seiner Konfiguration gezeigt.

Die einzelnen Implementierungen stellen selbst öffentliche Methoden bereit, damit ihre Eingaben gelesen und gesetzt werden können.

Durch die Supervisor View ist **M7: Graphische Benutzeroberfläche für Wahlleiter** erfüllt.

4.3 VoterView

Die Voter-View hat ähnlich zu der Supervisor-View eine zentrale Klasse, *VoterGUI*. Sie kann verschiedene Oberflächen anzeigen, die den Zustand der GUI repräsentieren. Diese Verschiedenen Oberflächen erweitern die Klasse *VoterGUIPanel* und überschreiben die öffentlichen Methoden, wenn sie diese durch die Methoden ihre Oberfläche anpassen (Bsp. Textfeld füllen).

Für die *VoterGUI* existieren folgende mögliche Zustände:

- **VoterAuthentication**: Der Wähler kann mithilfe der Oberfläche sein Zertifikat auswählen. Dieses Zertifikat wird an das *VoterSDKInterfaceImpl* gegeben und der Wähler authentifiziert. Dies erfüllt **F10: Authentifizierung mittels Zertifikat** aus Sicht der Benutzeroberfläche.

- **VoterChoice**: Dem Wähler werden die Kandidaten aufgelistet und er kann seine Stimme festlegen. Die *VoterChoice* wird abhängig von dem Wahlsystem mithilfe der *VoterVSComponentManager* Klasse konstruiert. Die die Methode createVotingForm() zur Verfügung stellt mit der das Wahlmenü erstellt wird. Hiermit ist **F11: Übermittlung der Stimme** aus Sicht der Benutzeroberfläche modelliert. Schließlich kann der Wähler über eine Schaltfläche seine Stimme endgültig abgeben.

- **VoterWait**: Solange die Wahl nicht beendet ist, aber der Wähler schon seine Stimme abgeben hat, wird ihm nicht das endgültige Ergebnis angezeigt. In diesem Zeitraum wird dem Wähler über die Benutzeroberfläche seine Stimmabgabe angezeigt und er wird darüber informiert, wann die Wahl spätestens vorbei ist.

- **VoterResult**: Das Ergebnis der Wahl wird in einem Diagramm dargestellt und die Stimme des Wählers angezeigt. Eine Schaltfläche lässt ihn seine Konfiguration bestätigen, was dazu führt das mit der Konfiguration eine Wahl aufgesetzt wird und somit **F8: Aktivierung der Wahl** aus Sicht der Benutzeroberfläche erfüllt.

Die Generierung der Diagramme und der Stimme geschieht über ein Objekt der *VoterVSComponentManager* Klasse, welche das richtige Diagramm und Wahlformular, abhängig vom Wahlsystem generiert. Die Schnittstellen von Control und Model werden wie in der Supervisor View in einer Adapter Klasse zusammengefasst. Die Klasse *VoterAdapter* kann Methoden in den beiden Interfaces *VoterViewToControlIF* und *VoterViewToModelIF* aufrufen. Ein Objekt dieser Klasse wird an alle Voter-View-Klassen überreicht, die auf die Schnittstellen zugreifen müssen.

Durch die Voter View ist **M8: Graphische Benutzeroberfläche für Wähler** erfüllt.

5 Control

Der grundlegende Aufbau der Control-Subpakete ist eine Control-Klasse, welche das entsprechende ViewToControlIF implementiert. Diese Implementierung hält alle relevanten Listener, welche durch die [*Supervisor*|*Voter*]*ViewToControlIF* gefordert sind und durch die Klasse [*Supervisor*|*Voter*]*EventListener* generalisiert sind. Diese Klasse implementiert *ActionListener* und hat Zugriff auf das Model- und View-Interface, damit es die vom Listener abhängigen Zustandsänderungen in der View oder Model anfordern kann. Jeder Listener wird einer oder mehreren *JButton*-Schaltflächen als *ActionListener* übergeben.

Für den Wahlleiter werden folgende Listener ausgelöst, wenn:

- **NewConfigListener**: der Wahlleiter eine neue Wahl konfigurieren möchte.
- **ImportConfigListener**: der Wahlleiter eine gespeicherte Wahlkonfiguration laden möchte.
- **ExportConfigListener**: der Wahlleiter eine Wahlkonfiguration abspeichern möchte.
- **ConfirmedConfigListener**: der Wahlleiter eine Wahl vollständig konfiguriert hat. Die Konfiguration wird daraufhin auf mögliche Probleme überprüft.
- **FinishElectionListener**: die derzeit aktive Wahl endgültig beendet werden soll.
- **FirstAuthenticationListener**: der Wahlleiter sich das erste Mal in dem Netzwerk anmeldet.
- **SupervisorAuthenticationListener**: der Wahlleiter sich mit seinem Zertifikat anmeldet.
- **SupervisorLogoutListener**: der Wähler seinen Klient beendet.
- **StartElectionListener**: eine Wahl aktiviert werden soll.

Für den Wähler werden folgende Listener ausgelöst, wenn:

- **VoterLogout**: der Wähler seinen Klienten beendet.
- **VoterAuthenticationListener**: der Wähler mit seinem Zertifikat anmeldet.
- **VotedListener**: der Wähler seine Stimme abgeben will.

6 Exceptions

- **AuthenticationException** Das angegebene Zertifikat ist invalide oder hat mangelnde Berechtigungen.
- **ConfigImportException** Die angegebene Datei enthält eine ungültige Wahlkonfiguration.
- **EnrollmentException** Beim Registrieren des Wählers ist ein Fehler aufgetreten.
- **ElectionCreationException** Beim Erstellen der Wahl ist ein Fehler aufgetreten.
- **ConnectionException** Es konnte keine Verbindung zum Blockchain-Netzwerk hergestellt werden.
- **VoteDeserializationException** Der angegebene JSON-String konnte nicht zu einem Vote deserialisiert werden.

7 Einteilung zur Implementierung

7.1 Abhängigkeiten

Das Control- und View-Paket kann jeweils unabhängig voneinander und unabhängig vom Model implementiert werden. Die Implementierung des StateManagements, der Transactions und des Chaincodes ist auch von keinen anderen Komponenten abhängig und kann somit direkt durchgeführt werden. Da die Supervisor- als auch die VoterView Tabellen und Diagramme benötigen, ist deren Implementierung vorher notwendig. Die Implementierung der SDKConnection basiert zwar auf Transactions, ist aber größtenteils unabhängig möglich, erst zu Integrationstests muss Transactions fertiggestellt sein.

7.2 Gantt-Diagramm

Die Implementierung verläuft, durch den vorgegebenen Zeitrahmen bedingt, größtenteils gleichzeitig. Das Ziel ist eine Just-In-Time-Fertigstellung der verschiedenen Module, d.h. zwei voneinander abhängige Module werden zu nahen Zeitpunkten fertiggestellt. Dies ermöglicht das frühe Zusammenfügen und Testen von größeren Zusammenhangskomponenten. Das Gantt-Diagramm in Abb. 15 stellt den groben Verlauf der nächsten Phase

dar, die hier gelisteten Personen implementieren aber nicht alleinig, sondern sind die Verantworlichen für das zugehörige Modul.

Hauptpaket	Einheiten	Verantwortliche	Dauer in Tagen
Schnittstellen		Tim	
Model		David	
	SDKConnection	Philipp	
	Transactions	David	
	StateManagement	David	
	Wahlsysteme	Artem/David	
View		Achim	
	VoterView	Achim	
	SupervisorDashbrd.	Achim	
	Konfigurationsmenü	Achim	
	Diagramme	Achim	
	Tabellen	Achim	
Control		Artem	
	SupervisorControl	Artem	
	VoterControl	Artem	
Chaincode		Tim	
	Metadaten Output	Tim	
	Metadaten Input	Tim	
	Wahlende-Prüfung	Tim	
	Stimmabgabe	Philipp	
	Neustart des Netzw.	Philipp	

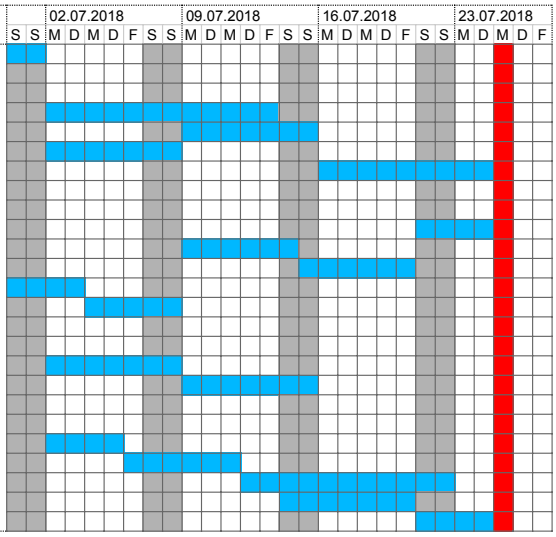

Abbildung 15: Gantt-Diagramm

8 Änderung zum Pflichtenheft

8.1 K2: Geheime Wahlen

Die Implementierung von geheimen Wahlen zeigte sich bei unseren Nachforschungen als besonders aufwändig, da der Ledger grundsätzlich „by design“ öffentlich ist. Mögliche Strategien wie das Verschlüsseln der Stimmen beim Wähler und späteres Entschlüsseln wurden als Lösung in Betracht gezogen. Selbst hiermit erweist sich die Umsetzung als problematisch, da die Auswertung der Stimmen vom Klienten in die Smart Contracts verlegt werden müsste, um wahrhaft geheim zu sein. Eine solche Auszählung wird durch die verschiedenen Wahlsysteme und deren unterschiedliche Auszählungsverfahren zusätzlich erschwert. Sie wäre in einem solchen System selbst für einen technikaffinen Wähler undurchsichtig, weil nur schwer nachzuverfolgen ist, welcher Chaincode tatsächlich eingesetzt wird. Im Gegensatz dazu könnte beispielsweise der Source Code des Klienten offengelegt werden und ein solcher Wähler könnte nachvollziehen, dass die Auswertung korrekt vonstatten ging.
Wir schließen deshalb die Implementierung von geheimen Wahlen nicht aus, bezweifeln aber deren Umsetzbarkeit durch uns. Sollte sich das zu einem späteren Zeitpunkt ändern, ziehen wir sie dennoch in Betracht.