KIT – Karlsruhe Institut of Technology

Praxis der Software-Entwicklung WS 2009/10

Gruppe 7

»Entwurfsdokument« v 1.0

Analysesoftware für Soziale Netzwerke

19.12.2009

Phase	Phasenverantwortlich	e-mail
Pflichtenheft	Mark Engel	mark.engel@student.kit.edu
Entwurf	Felix Stahlberg	felix.stahlberg@student.kit.edu
Implementierung	Marc Typke	marc.typke@student.kit.edu
Testen	Felix Stahlberg	felix.stahlberg@student.kit.edu
Präsentation	Mark Engel	mark.engel@student.kit.edu

Inhaltverzeichnis

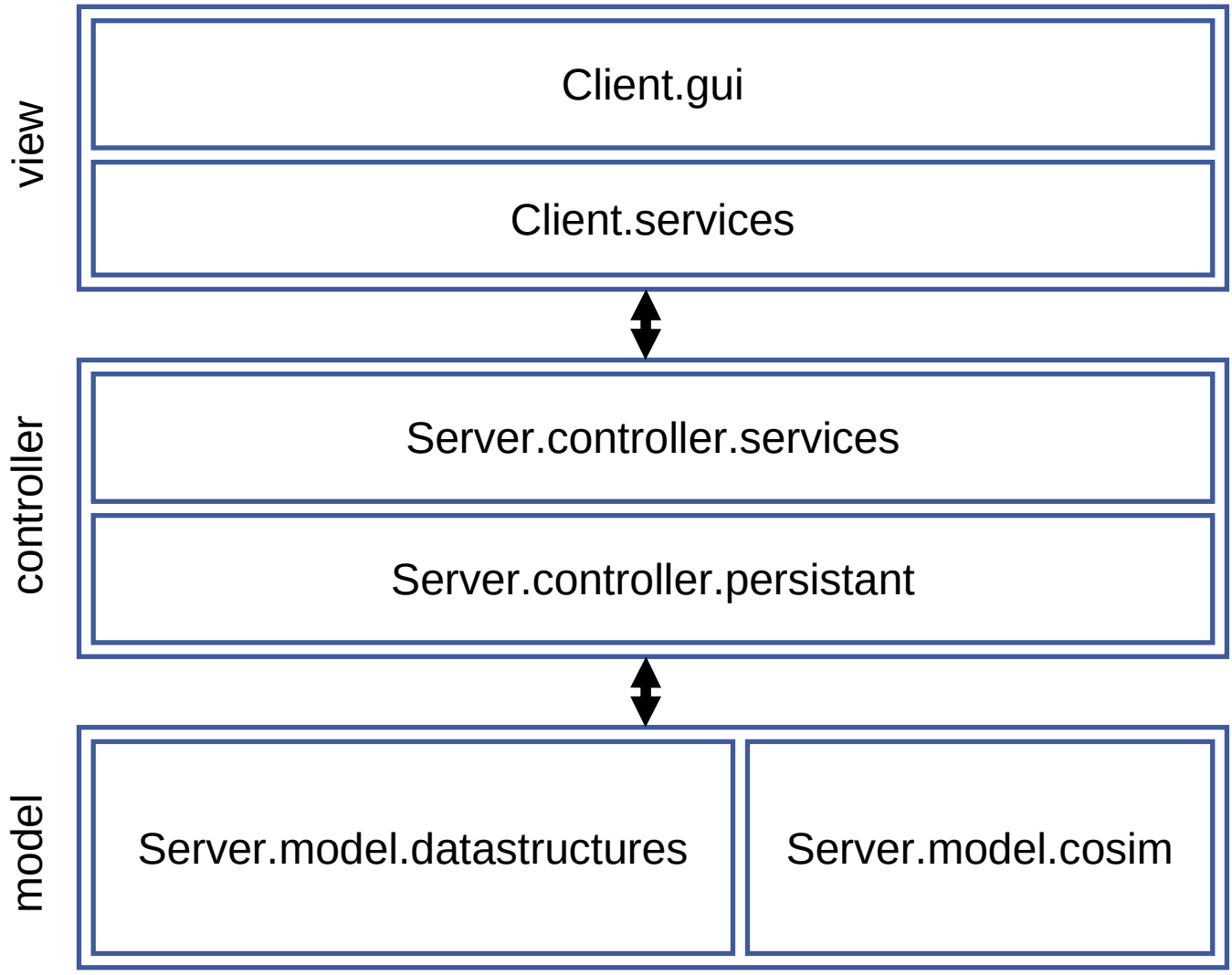
1 Einleitung

Dieses Dokument definiert die Systemarchitektur der Analysesoftware SocialView und detailliert die Beziehungen und Abhängigkeiten sowie die Interaktion der einzelnen Klassen untereinander. Diese werden in Klassendiagrammen und einigen Sequenzdiagrammen visualisiert. Der Punkt Datenhaltung wird nochmals gesondert mittels ER-Diagrammen betrachtet.

Der Entwurf setzt die Verwendung des Google Web Toolkit 2.0 (GWT) voraus.

2 Aufbau

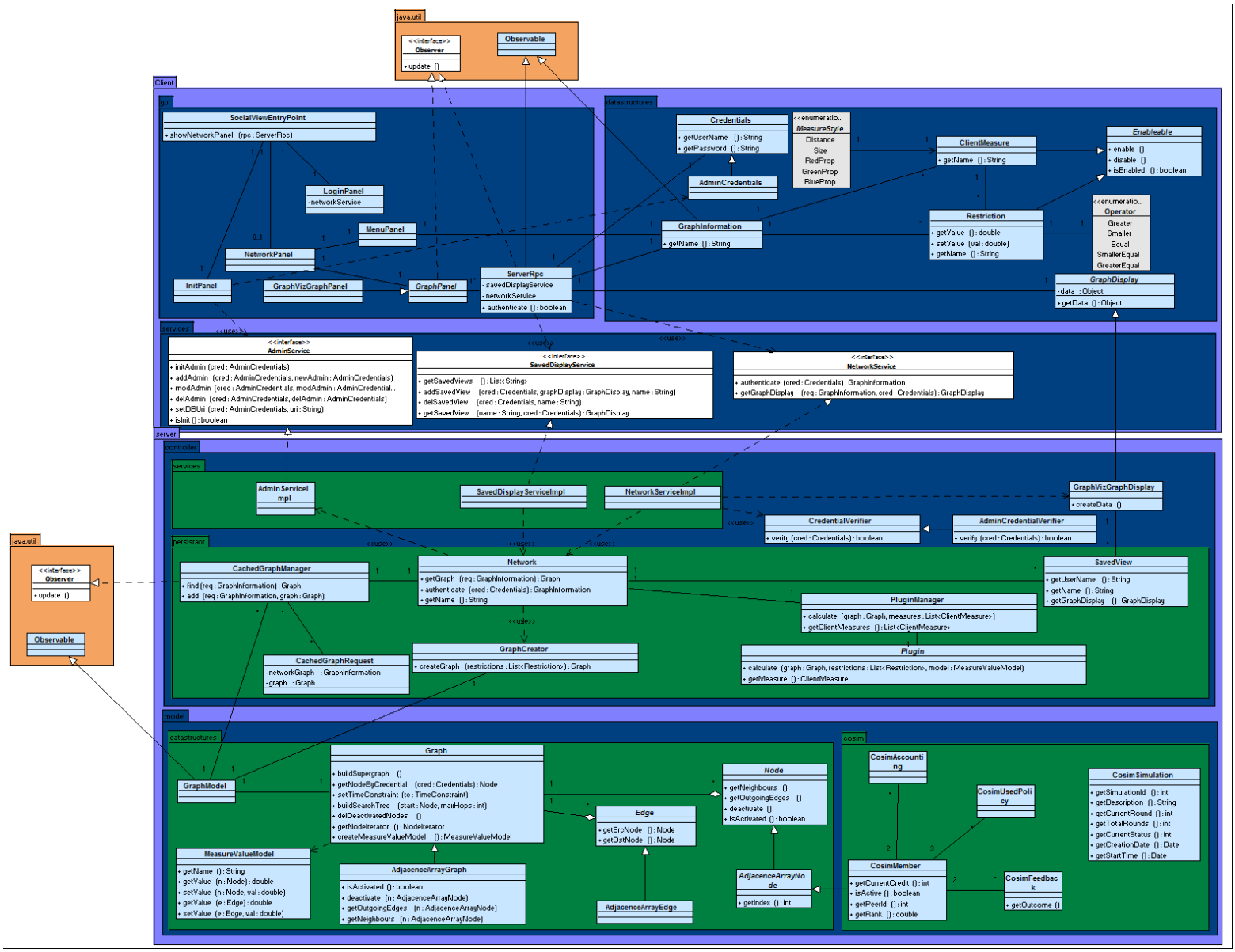
2.1 Systemarchitektur



Die Software strukturiert sich durch eine intransparente 3-Schichten-Architektur (Benutzeroberfläche, Anwendungsschicht und Datenbankschicht), bei der die einzelnen Schichten teilweise intern feiner partitioniert sind. Kommunikation zwischen Schichten geschieht also nur entweder zwischen Benutzeroberfläche und Anwendungsschicht oder zwischen Anwendungsschicht und Datenbanksschicht. Die drei Schichten übernehmen grob auch die einzelnen Aufgaben einer MVC-Architektur (die Benutzeroberfläche die des Views, die Anwendungsschicht die des Controllers und die Datenbanksschicht die des Models), jedoch mit dem Unterschied, dass View und Model in keiner Beziehung untereinander stehen. Dies hat den Vorteil der besseren Austauschbarkeit der beiden Schichten (da nur eine, nicht zwei Schnittstellen bedient werden müssen, die teilweise noch voneinander abhängen). Besonders die Austauschbarkeit / Erweiterbarkeit des Modells ist in diesem Projekt besonders gewünscht, da es auf lediglich ein soziales Netzwerk angepasst ist.

Das Architekturmodell wird durch die vorgegebene Anatomie von GWT-Projekten teilweise ein wenig aufgeweicht.

2.2 Klassendiagramm



3 Klassenbeschreibungen

3.1 Übersicht

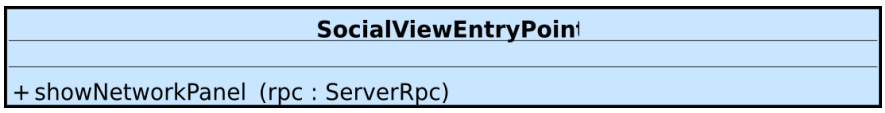
GWT schreibt die Strukturierung des Projektes in das client und das server Packages vor, wobei im Gegensatz zu Code im server Package der Code in client in JavaScript übersetzt wird und auf dem Clientrechner im Browser läuft. Das zieht technische Konsequenzen nach sich.

3.2 Client

3.2.1 Client GUI (client.gui)

Dieses Paket hat eine starke Bindung zu GWT. Hier liegen alle Klassen der Benutzeroberfläche. Es stellt also die oberste Schicht der Schichten-Architektur dar. Die Benutzeroberfläche ist mit einem EntryPoint und verschiedenen Panels, die jeweils spezifische Aufgaben haben, organisiert.

3.2.1.1 Die Klasse SocialViewEntryPoint



SocialViewEntryPoint ist die Eintrittsklasse. In dieser Klasse beginnt die Interaktion des Users mit dem Programm.

Methoden

- *public void onModuleLoad()* Die Implementierung dieser Funktion wird durch das Interface EntryPoint (GWT) vorgegeben. Sie wird bei der Initalisierung des Moduls aufgerufen und stellt somit eine Methode ähnlich der Main-Methode bei Standard Java-Anwendungen dar. Hier wird das Panel LoginPanel erstellt und der Benutzeroberfläche hinzugefügt.
- *public void showNetworkPanel(ServerRpc rpc)* Diese Funktion wird vom LoginPanel nach erfolgreichem Login aufgerufen. Es wird das LoginPanel von der Benutzeroberfläche entfernt und dafür ein NetworkPanel erstellt und hinzugefügt.

3.2.1.2 Die Klasse InitPanel

Falls noch keine Datenbank-Verbindung und kein Admin konfiguriert wurden, wird dieses Panel aufgerufen. In diesem Panel kann die initiale Konfiguration des Admins und der Verbindungsdaten über das Webinterface konfiguriert werden.

3.2.1.3 Die Klasse LoginPanel

Das LoginPanel ermöglicht dem User seine Login Daten einzugeben um sich zu authentifizieren. Dafür erzeugt LoginPanel eine ServerRpc-Instanz mit den eingegebenen Credentials und ruft auf dieser Instanz dann authenticate auf. Wird true zurückgegeben war die Authentifizierung erfolgreich und entryPoint.showNetworkPanel() wird aufgerufen.

Attribute

- *private EntryPoint entryPoint* Speichert einen Verweis zum EntryPoint

Methoden

- *public LoginPanel(EntryPoint entryPoint)* Einziger Konstruktor. Ein Verweis zu dem EntryPoint muss mit übergeben werden, damit nach einem erfolgreichen Login SocialViewEntryPoint.showNetworkPanel() aufgerufen werden kann.

3.2.1.4 Die Klasse NetworkPanel

Dieses Panel wird angezeigt, nachdem sich der User bei einem Netzwerk eingeloggt hat. Es beinhaltet als Leiste ein MenuPanel und einGraphPanel in dem der Graph visualisiert wird.

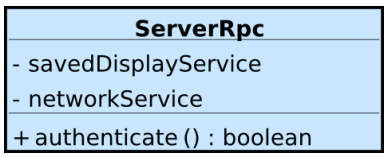
Attribute

- *private GraphPanel graphPane*l Das GraphPanel, das auf dem NetworkPanel angezeigt wird
- *private MenuPanel menuPanel* Das MenuPanel, das auf dem NetworkPanel angezeigt wird

Methoden

- *public NetworkPanel(ServerRpc rpc)* Einziger Konstruktor. Erstellt ein GraphPanel und ein MenuPanel und fügt diese sich selbst zu. Eine ServerRpc-Instanz muss mit übergeben werden, damit diese bei der Konstruktion des GraphPanels weitergereicht werden kann.

3.2.1.5 Die Klasse ServerRpc



Die Klasse ServerRpc kapselt die Kommunikation des Clients mit dem Server. Des Weiteren beobachtet ServerRpc eine GraphInformation-Instanz, nachdem sie in authenticate() erstellt wurde. So wirdServerRpc benachrichtigt, sofern der User über MenuPanel die Graphspezifikationen angepasst hat aktualisiert GraphDisplay und benachrichtigt dann eigene Beobachter wie das GraphPanel.

Attribute

- *private NetworkServiceAsync networkService* Proxy zur Kommunikation mit dem Server via der Schnittstelle NetworkService
- *private SavedViewsServiceAsync savedViewsService* Proxy zur Kommunikation mit dem Server via der Schnittstelle SavedViewsService
- *private GraphDisplay graphDisplay* GraphDisplay, das den aktuell angezeigten Graphen charakterisiert. null sofern kein Graph angezeigt wird (z.B. wenn der User noch nicht eingeloggt ist). DasGraphDisplay wird von einem GraphPanel beobachtet, das bei Änderungen informiert wird und die Anzeige aktualisiert.
- *private GraphInformation graphInformation* Informationen zum aktuell angezeigten Graphen, null sofern der User noch nicht authentifiziert ist (noch nicht authenticate() aufgerufen wurde)
- *private Credentials cred* Credentials zur Authentifizierung bei der Kommunikation mit dem Server.

Methoden

- *public ServerRpc(String userName, String pass)* Einziger Konstruktor.
- *public boolean authenticate()* Überprüft mittels networkService.authenticate(), ob die gespeicherten Credentials valide sind. Sofern dies der Fall ist wird eineGraphInformation-Instanz zurückgegeben die dann in graphInformation abgelegt wird und zusätzlich beobachtet wird.
- *public void update(Observable o, Object arg)* Implementierung des Interfaces java.util.Observer. Das ServerRpc wird bei Änderungen an dem GraphInformation mittels dieser Methode benachrichtigt, um dann mit dem Server mittels der Schnittstelle NetworkService.getGraphDisplay zu kommunizieren und einen aktualisierten GraphDisplay zu erhalten. Anschließend werden darüber mit notifyObservers() Beobachter wie GraphPanel informiert.
- *public GraphInformation getGraphInformation()* Gibt beobachtete GraphInformation

3.2.1.6 Die abstrakte Klasse GraphPanel

In diesem Panel wird der Graph angezeigt. Es beobachtet einen NetworkRpc und aktualisiert die Anzeige, sobald sich dessen assozierter GraphDisplay verändert. Diese Klasse ist abstrakt, da es für die verwendete Graphdarstellungsbibliothek eine konkret implementierende Unterklasse geben muss.

Methoden

- *public GraphPanel(ServerRpc rpc)* Registriert sich als Observer bei dem übergebenen ServerRpc
- *abstract public void update(Observable o, Object arg)* Implementierung des Interfaces java.util.Observer. Sobald das beobachtete ServerRpc die Beobachter benachrichtigt steht ein aktualisierter GraphDisplay zur Verfügung, das hier verwendet werden kann um die Anzeige zu aktualisieren.. Diese Methode ist abstrakt, da es von der konkret verwendeten

Graphdarstellungsbibliothek abhängt, wie das GraphDisplay genau zu verarbeiten ist.

3.2.1.7 Die Klasse GraphVizGraphPanel

Diese Klasse ist die Implementierung von GraphPanel für die Graphendarstellungsbibliothek GraphViz.

Methoden

- *abstract public void update(Observable o, Object arg*) Implementierung des Interfaces java.util.Observer. Das GraphDisplay im beobachteten GraphDisplayModel müssen vom Typ GraphVizGraphDisplay sein. GraphVizGraphDisplay.deploy() gibt dann ein xdot-Source als String zurück, der dann an die JavaScript CanViz übergeben und somit der Graph angezeigt werden kann.

3.2.1.8 Die Klasse MenuPanel

Dieses Panel ermöglicht dem Nutzer den angezeigten Graphen durch die Aktivierung und Anpassungen von Zentralitätsmaßen und Restriktionen anzupassen. Ist der Benutzer mit den Anpassungen fertig und möchte den Graphen angezeigt bekommen, so greift MenuPanel auf die assozierte GraphInformation zu und verändert die ClientMeasure- und Restriction-Instanzen entsprechend. Anschließend wird GraphInformation.setChanged() und GraphInformation.notifyObservers() aufgerufen, um das ServerRpc als Observer von dem GraphInformation zu benachrichtigen, damit dieses das assozierte GraphDisplay aktualisieren kann und wiederum seine Observer benachrichtigen kann.

Attribute

- *private GraphInformation gi* Die GraphInformation, der verändert wird, wenn der Benutzer Restrictions verändert oder Zentralitätsmaße (de)aktiviert.

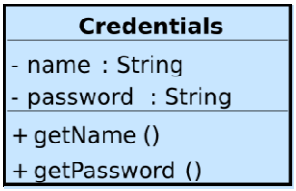
Methoden

- *public MenuPanel(NetworkGraph ng)* Einziger Konstruktor.

3.2.2 Client Datenstrukturen (client.datastructures)

Hier werden Datenstrukturen definiert, die im Client benötigt werden und in JavaScript übersetzt werden müssen. Es handelt sich hierbei vor Allem um Datenstrukturen, die zwischen Server und Client verschickt werden.

3.2.2.1 Die Klasse Credentials



In Credentials werden Logininformationen gekapselt, um sie an den Server zu schicken und sich zu authentifizieren. Außerdem dient diese Klasse zur Verifizierung des Users bei jeder Serveranfrage.

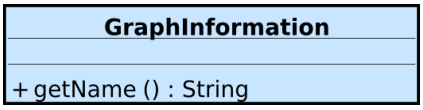
Attribute

- *private String userName* Der Name des Users.
- *private String password* Das Passwort, verschlüsselt mittels dem Message-Digest Algorithm 5 (MD5).

Methoden

- *public Credentials(String name, String pass)* Einziger Konstruktor
- *public String getUserName()* Gibt den Usernamen zurück
- *public String getPassword()* GIbt das Passwort zurück

3.2.2.2 Die Klasse GraphInformation



Diese Klasse repräsentiert das soziale Netzwerk im Clienten. Beim Login des Users wird sie einmal im Server instanziert und dem Client geschickt (über ServerRpc.authenticate()), dem somit alle nötigen Informationen verfügbar sind, die für den Aufbau vonNetworkPanels nötig sind. Dazu gehören zum Beispiel Listen verfügbarer Zentralitätsmaße und möglicher Restriktionen. Zudem können über diese Klasse durch Aktivieren oder Verändern assozierter Restriktionen und ZentralitätsmaßeGraphenanfragen spezifiziert werden. Zusätzlich erbt GraphInformation von java.util.Observable, da das ServerRpc NetworkGraphs beobachtet, um bei Änderungen die Aktualisierung der Anzeige einzuleiten. Zu beachten ist jedoch, dass die Benachrichtigungen der Observer (GraphInformation.notifyObservers()) nur vom MenuPanel angestoßen wird.

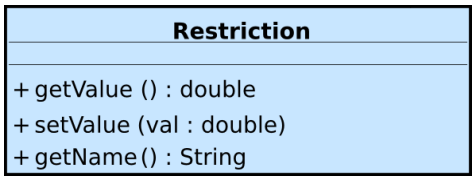
Attribute

- *private String name* Name des Netzwerkes
- *private List<Restriction> restrictions* Mögliche globale Beschränkungen
- *private List<ClientMeasure> measures* Die verfügbaren Zentralitätsmaße

Methoden

- *public String getName()* gibt den Namen des Netzwerkes aus
- *public List<Restriction> getRestrictions()* gibt alle globalen Beschränkungen zurück
- *public List<ClientMeasure> getMeasures()* gibt alle vorhandenen Maße zurück

3.2.2.3 Die Klasse Restriction



Die Restriction Klasse ermöglicht es, Graphanfragen feiner zu spezifizieren und einzuschränken. Sie kann zum Einen dazu verwendet werden, globale Begrenzungen zu definieren (Zum Beispiel: Zeige höchstens 50 Knoten an). In diesem Fall existiert ein Verweis auf die Restriction in GraphInformation. Wird stattdessen die Restriction in einemClientMeasure referenziert, werden hier Zentralitätsmaße beschränkt (zum Beispiel User A und B müssen sich gegenseitig mindestens drei Nachrichten geschickt haben um als Knoten im Graph repräsentiert zu werden).

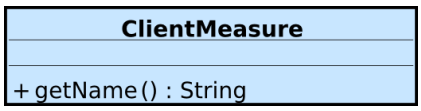
Attribute

- *private double value* Der Zahlenwert, den diese Restriction hat. Dieser Zahlenwert wird bei der Anwendung der Restriction mittels dem gegebenen Operator mit dem tatsächlichen Wert des Zentralitätsmaßes verknüpft.
- *private Operator operator* Der zu verwendende binärer Vergleichsoperator
- *private String name* Der Name der Restriction

Methoden

- *public void setValue()* Ändert den Zahlenwert der Restriction auf diesen Wert
- *public double getValue()* Gibt den momentanen Zahlenwert aus
- *public Operator getOperator()* Gibt den verwendeten Operator zurück.
- *public void setOperator(Operator op)* Setzt den zu verwendenden Operator
- *public String getName()* Gibt den Namen der Restriction zurück.
- *public Restriction(String name)* Einziger Konstruktor

3.2.2.4 Die Klasse ClientMeasure



Diese Klasse enthält Informationen zu einem Zentralitätsmaß, sodass der Anwender dieses Maß im Browser auswählen kann und veränderte Instanzen wieder an den Server schicken kann, damit dieser den passenden Graphen berechnet.

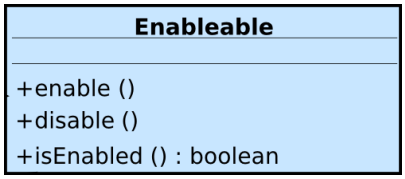
Attribute

- *private String name* Name des Zenträlitätsmaßes
- *private MeasureStyle style* Die aktuelle Anzeigeart
- *private List<Restriction> restrictions* Eine Liste von Restrictions die für das aktuelle Zentralitätsmaß verwendet werden

Methoden

- *MeasureStyle getStyle()* gibt das aktuelle verwendete MeasureStyle zurück
- *void setStyle(MeasureStyle style)* setzt das aktuelle MeasureStyle
- *List<Restriction> getRestrictions()* gibt eine Liste der für dieses Measure verwendeten Restrictions

3.2.2.5 Die abstrakte Klasse Enableable



Instanzen von Unterklassen von Enableable können deaktiviert und aktiviert werden. Von dieser Klasse erben zum Beispiel ClientMeasure und Restriction. Somit kann, wenn der NetworkGraph zurück an den Server geschickt wird, der Server überprüfen, ob und welche Maße und Restrictionen vom User erwünscht sind.

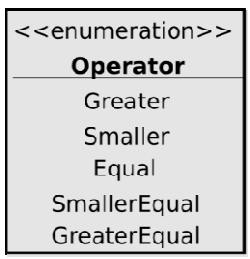
Attribute

- *private boolean enabled* Signalisiert ob die Instanz aktiviert ist

Methoden

- *public void enable()* Aktiviert die Instanz
- *public void disable()* Deaktiviert die Instanz
- *public boolean isEnabled()* Gibt die aktuellen Zustand aus

3.2.2.6 Die Klasse Operator (enum)

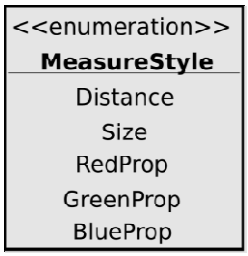


Diese Enumeration Klasse enthält Vergleichsoperatoren zur Benutzung mit den Restrictions

Elemente

- *Smaller* Kleiner-Als-Operator
- *SmallerEqual* Kleiner-Gleich-Operator
- *Greater* Größer-Als-Operator
- *GreaterEqual* Größer-Gleich-Operator
- *Equal* Gleich-Operator

3.2.2.7 Die Klasse MeasureStyle (enum)

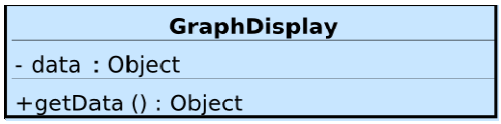


Diese Klasse enthält eine Auswahl wie die Measures später im Graphen angezeigt werden sollen.

Elemente

- *Distance* Darstellung durch Entfernung zwischen den Knoten
- *Size* Darstellung durch Größe der Knoten oder Kanten
- *RedProp* Darstellung durch die Stärke des Rotanteils des Elements
- *GreenProp* Darstellung durch die Stärke des Grünanteils des Elements
- *BlueProp* Darstellung durch die Stärke des Blauanteils des Elements

3.2.2.8 Die Klasse GraphDisplay



Eine Instanz der Klasse GraphDisplay repräsentiert eine vollständige Beschreibung der Darstellung eines konkreten Graphen. Diese Beschreibung ist abhängig von der verwendeten Graphendarstellungsbibliothek. Eine Implementierung gemäß dieser Graphendarstellungsbibliothek muss auf dem Server existieren. Somit übernimmt diese Klasse grob die Funktion eines Proxys im Client für GraphVizGraphDisplay auf dem Server.

Attribute

- *private Object data* Vollständige Daten, die für die Graphendarstellungsbibliothek nötig sind, um den Graphen anzeigen zu können

Methoden

- *public Object getData()* Gibt gespeicherte Daten zurück.

3.2.3 Client Services (client.services)

Die hier definierten Schnittstellen stellt der Server dem Client bereit. Die tatsächliche Kommunikation geschieht mittels Proxys auf von GWT vorgegebenen Wege. Die gesamte Serverfunktionalität wird in zwei Fassaden gekapselt.

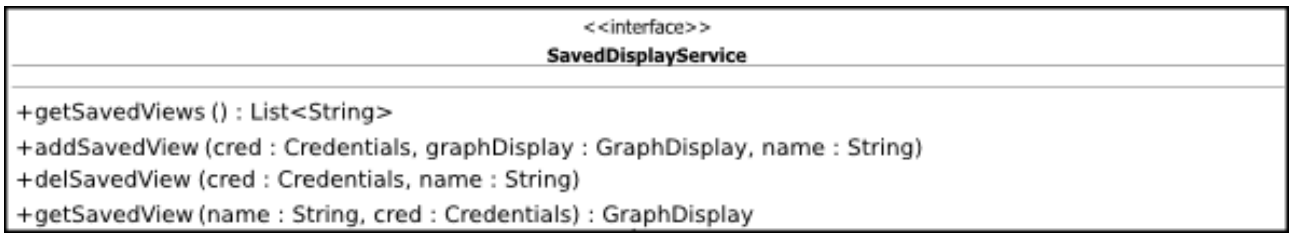
3.2.3.1 Das Interface NetworkService

Mit NetworkService kann der Client sich beim Server authentifizieren, Informationen über das Netzwerk erfahren und Graphenanfragen schicken sowie Graphendarstellungen erhalten.

Methoden

- *GraphInformation authenticate(Credential cred)* Nach einer erfolgreichen Authentication erhält der User ein GraphInformation Objekt, was alle relevanten Informationen über das Netzwerk enthält.
- *GraphDisplay getGraphDisplay(GraphInformation gi, Credential cred)* Mit diesem Interface werden Graphenanfragen an den Server geschickt. Vorrausgesetzt, dass die mitgesendeten Credentials valide sind, bekommt der Client einen GraphDisplay zurückgesendet, den der GraphPanel anzeigen kann.

3.2.3.2 Das Interface SavedDisplayService

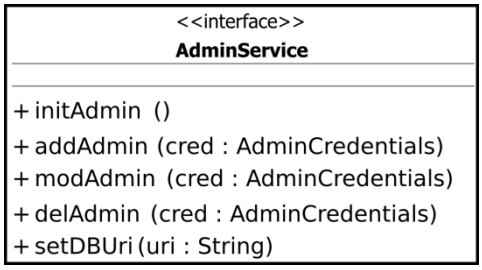


Mit diesem Interface, was optional als Wunschkriterium implementiert wird, können modifizierte Graphenansichten auf dem Server gespeichert und zu einem späteren Zeitpunkt wieder geladen werden.

Methoden

- *List<String> getSavedViews(Credential cred)* Gibt eine Liste aller gespeicherten Graphenansichten des jeweiligen Users zurück
- *void addSavedView(Credential cred, GraphDisplay graphDisplay, String name)* Fügt eine weitere Graphenansicht hinzu.
- *void delSavedView(Credential crendetial, String name)* Löscht eine Graphenansicht aus der Datenbank des Servers.
- *GraphDisplay getSavedView(Credential crendetial, String name)* Holt einen gespeicherten SavedView vom Server.

3.2.3.3 Das Interface AdminService



In diesem Service kann der Zugang zur Datenbank eingestellt und Administratoren verwaltet werden.

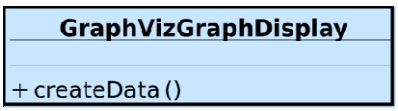
Methoden

- *void initAdmin(AdminCredential cred)* Falls noch kein Admin definiert ist, kann hiermit ein Admin erstellt werden
- *void addAdmin(AdminCredential cred, AdminCredential newAdmin*) Fügt einen weiteren Admin der Datenbank hinzu
- *void modAdmin(AdminCredential cred, AdminCredential modAdmin)* Modifiziert die Daten eines Admins
- *void delAdmin(AdminCredential cred, AdminCredential delAdmin)* Löscht einen Admin aus der Datenbank
- *boolean isInit()* Gibt zurück, ob das Spiel bereits initialisiert wurde und benutzbar ist

3.3 Controller (Server)

Dieses Paket enthält die mittlere Schicht der Schichtenarchitektur. Die Anwendungsschicht kapselt die Anwendungslogik und bildet das Verbindungsstück zwischen Benutzeroberfläche und Datenhaltungsschicht.

3.3.1 Die Klasse GraphVizGraphDisplay



GraphVizGraphDisplay ist die Implementierung von GraphDisplay. Diese Implementierung ist auf die Graphendarstellungsbibliothek GraphViz zugeschnitten.

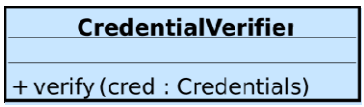
Attribute

- *private Object data xdot* Source als String, sobald createData() aufgerufen wurde

Methoden

- *public Object getData()* Gibt den xdot Source zurück
- *public createData(Graph graph, GraphInformation gi)* Erstellt aus einem Graph und den entsprechenden Informationen in GraphInformation xdot Source und speichert diese ab, sodass dieses Objekt anschließend zum Client verschickt werden kann.

3.3.2 Die Klasse CredentialsVerifier



Diese Klasse überprüft übergebene Credentials auf ihre Richtigkeit .

Methoden

- *public static boolean verify(Credentials c)* Diese Funktion überprüft, ob die aktuellen Credentials das Recht haben, auf das Netzwerk zuzugreifen. Übergebene Credentials sind gültig, sofern sie eine Entsprechung in der Cosim-Datenbanktabelle User haben.

3.3.3 Die Klasse AdminCredentialsVerifier

Eine Erweiterung der CredentialVerifier Klasse. Hier wird nicht nur auf User Credentials überprüft. Sondern auch kontrolliert, ob die übergebenen Credentials die Zugangsdaten eines Administrators sind.

3.3.4 Services (server.controller.services)

Alle Interfaces in client.services besitzen eine Implementierung in server.controller.services, die dann als Proxy referenziert werden können. Da es sich hier lediglich um Fassaden handelt, haben Klassen in diesem Package keinen funktionalen sondern nur deligierenden Charakter.

3.3.4.1 Die Klasse SavedDisplayServiceImpl

SavedDisplayServiceImpl stellt die serverseitige Implementierung der in client.services definierten SavedDisplayService Schnittstelle dar. Nachdem die übergebenen Credentials mittels CredentialVerifier verifiziert sind werden alle Anfragen werden an Network in server.controller.persistant deligiert.

3.3.4.2 Die Klasse NetworkServiceImpl

NetworkServiceImpl stellt die serverseitige Implementierung der in client.services definierten NetworkService Schnittstelle dar. Nachdem die übergebenen Credentials mittelsCredentialVerifier verifiziert sind werden alle Anfragen werden an Network in server.controller.persistant deligiert.

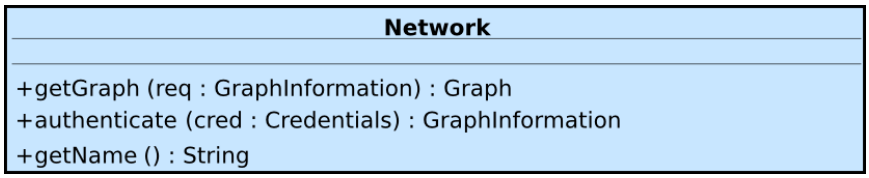
3.3.4.3 Die Klasse AdminServiceImpl

NetworkServiceImpl stellt die serverseitige Implementierung der in client.services definierten AdminService Schnittstelle dar.

3.3.5 Persistierungen (server.controller.persistant)

Alle Instanzen von Klassen in diesem Paket werden in der applikationseigene Datenbank mittels der JPA persistent gehalten. Weiterhin liegt hier die Anwendungslogik sowie Mechanismen zur Erweiterbarkeit der Software durch Plugins.

3.3.5.1 Die Klasse Network



Hier werden alle Informationen über das eingebundene soziale Netzwerk gespeichert. Die ganze Klasse Network wird persistent in einer Datenbank gespeichert, sodass auch nach einen Neustart des Servers alle Daten verfügbar sind.

Attribute

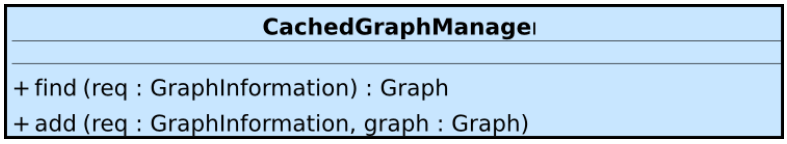
- *private String name* Der Netzwerknamen
- *private PluginManager* pluginManager

- *private CachedGraphManager cachedGraphManager* Kümmert sich um die Verwaltung der gecachten Graphenanfragen

Methoden

- *public Graph getGraph(GraphInformation req)* Empfängt einen Anfrage an einen Graphen und wird eine visualisierte Ausgabe zurückgeben.
- *public GraphInformation authenticate(Credentials cred)* Überprüft Credentials und liefert im Erfolgsfall eine Client-Repräsentation des Netzwerks zurück, das dann über NetworkService wieder an den Client zurückgeschickt wird.

3.3.5.2 Die Klasse CachedGraphManager

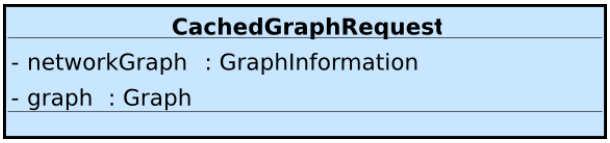


Diese Klasse kümmert sich um das Caching bereits berechneter Graphen. Außerdem beobachtet CachedGraphManager den Supergraphen und löscht bei Veränderungen veralterte Cacheeinträge.

Methoden

- *public Graph find(GraphInformation request)* Durchsucht seinen Cache nach einen passenden Eintrag
- *public void add(GraphInformation request, Graph graph)* Speichert den Graphen im Cache.
- *public void update(Observable o, Object arg*) Implementierung des Interfaces java.util.Observer. Aktualisiert den Cache bei verändertem Supergraphen (zum Beispiel bei neuen Runden).

3.3.5.3 Die Klasse CachedGraphRequest

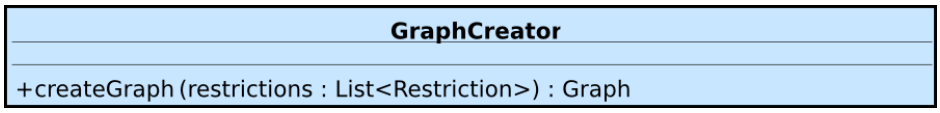


Speichert bereits angeforderte Graphen um performanter zu sein. Instanzen dieser Klasse werden ausschließlich in CachedGraphManager gehalten.

Attribute

- *private NetworkGraph networkGraph* Die gespeicherte Anfrage
- *private Graph graph* Der berechnete Graph

3.3.5.4 Die Klasse GraphCreator

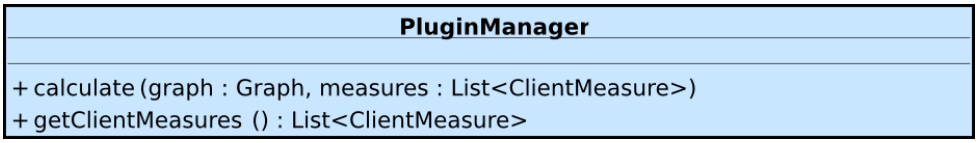


Diese Klasse für die Erstellung von Graphen zur Weiterverabeitung durch Plugins verantwortlich.

Methoden

- *public Graph createGraph(Set<Restrictions> restrictions)* Kopiert den Supergraphen, beschneidet ihn gemäß gegebener Restriktionen und gibt den beschnittenen Graphen zurück.

3.3.5.5 Die Klasse PluginManager

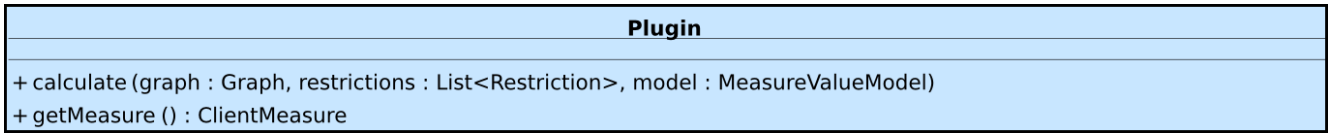


Der PluginManager übernimmt die Instanzierung und die Verwaltung der vorhandenen Plugins. Zudem überprüft der PluginManager regelmäßig die Existenz neuer Plugins und bindet diese gegebenenfalls ein.

Methoden

- *public PluginManager()* Instanziert alle existierenden Plugins und speichert Verweise auf sie.
- *public void calculate(Graph graph, List<ClientMeasures> measures)* Diese Methode stößt die Berechnung der Zentralitätsmaße durch die Plugins an. Dazu erstellt der PluginManager für jedes Plugin, das in measures aktiviert ist, mittels der Fabrikmethode Graph.createMeasureValueModel() ein eigenes Model, in das deren Berechnungen abgelegt werden können, und ruft Plugin.calculate() mit jeweiligen Models und den in measures definierten Restrictions für jedes Plugin das für ein aktiviertes Zentralitätsmaß verantwortlich ist, auf. Diese Plugins füllen dann ihre Models und deaktivieren Knoten, falls es die übergebenen Restrictions erfordern. Nachdem alle Plugins mit den Berechnungen fertig sind wird Graph.delDeactivatedNodes() aufgerufen.

3.3.5.6 Die abstrakte Klasse Plugin

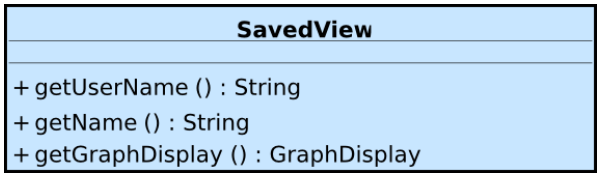


Diese Klasse stellt eine abstrakte Oberklasse der installierten Plugins dar. Ein Plugin implementiert immer genau ein Zentralitätsmaß.

Methoden

- *public Graph calculate(Graph graph, List<Restriction> restrictions, model MeasureValueModel)* Benutzt das model, um Berechnungen zu diesem Zentralitätsmaßen als Kanten- oder Knotengewichte abzulegen. Restrictions werden angewandt, indem Knoten deaktiviert werden.
- *public ClientMeasure getMeasure()* Gibt ein ClientMeasure zurück, das an den Clienten geschickt wird um die Auswahloptionen zu diesem Zentralitätsmaß grafisch darzustellen.

3.3.5.7 Die Klasse SavedDisplay



Diese Klasse enthält eine zu speichernde Graphenansicht eines Users.

Attribute

- *private String userName* Name des Users
- *private String name* Name der gespeicherten Ansicht
- *private GraphDisplay graph* Die gespeicherte Graphansicht

Methoden

- *public String getUserName()* Gibt den Usernamen zurück
- *public String getName()* Gibt den Namen der Ansicht zurück
- *public GraphDisplay getGraph()* gibt die gespeicherte Graphendarstellung zurück

3.4 Model (Server)

3.4.1 Cosim Datenbankklassen (server.model.cosim)

Klassen in diesem Paket werden auf Tabellen in der Cosim Datenbank, Instanzen dieser Klassen auf deren Zeilen abgebildet. Sie sind also Entities im Sinne von JPA.

3.4.1.1 Die Entity-Klasse CosimMembers

Abbildung der Tabelle SIMULATION_MEMBERS. Die Klasse erbt zusätzlich von AdjacenceArrayNode und bildet die Brücke zwischen Graphenrepräsentation und Datenbank.

Attribute

- *private int simulationId* Id der aktuellen Simulation
- *private int peerId* aktuelle PeerId
- *private int currentPolicyId* aktuelle PolicyId
- *private int currentQoSPolicyId* aktuelle QoSPolicyId
- *private int currentRecPolicyId* aktuelle RecPolicyId
- *private int currentCredit* aktuelles Guthaben
- *private int rank* aktuelle Platzierung
- *private int active* ob dieser Peer aktiv ist

Methoden

- *public CosimMember getPeer()* gibt den jeweiligen Peer zurück
- *public CosimUsedPolicy getCurrentPolicy()* gibt die aktuelle Policy aus
- *public CosimUsedPolicy getCurrentQoSPolicy()* gibt die aktuelle QoS Policy aus
- *public CosimUsedPolicy getCurrentRecPolicyId* gibt die aktuelle Rec Policy aus
- *public int getCurrentCredit()* gibt das aktuelle Guthaben aus
- *public int getRank()* gibt die aktuelle Platzierung aus
- *public int getActive* ist der Peer aktiv

3.4.1.2 Die Entity-Klasse CosimAccounting

Abbildung der Tabelle SIMULATION_ACCOUNTING

Attribute

- *private int simulationId* Id der aktuellen Simulation
- *private int roundId* Nummer der momentanen Runde
- *private int peerId* aktuelle PeerId
- *private int serviceProviderId* aktuelle Id des Service Providers
- *private int credit* aktuelles Guthaben
- *private int serviceProviderCredit* Guthaben des Service Providers

- *private int newCredit* neues Guthaben
- *private int newServiceProviderCredit* neues Guthaben des Service Providers
- *private int QoS* Quality of Service
- *private int serviceProviderQoS* Quality of Service des Service Providers
- *private int newQoS* die neue Quality of Service
- *private int newServiceProviderQoS* die neue Quality of Service des Service Providers
- *private int policyId* aktuelle Policy Id
- *private int serviceProviderPolicy* die Policy des ServiceProviders

Methoden

- *public int getRoundId* Gibt die jeweilige Rundennummer aus
- *public CosimMember getPeer()* gibt den jeweiligen Peer zurück
- *public CosimMember getServiceProvider()* gibt den jeweiligen Service Provider zurück
- *public int getCredit()* gibt das aktuelle Guthaben zurück
- *public int getServiceProviderCredit()* gibt das aktuelle Guthaben des Service Providers zurück
- *public int getNewCredit()* gibt das neue Guthaben zurück
- *public int getNewServiceProviderCredit()* gibt das neue Guthaben des Service Providers zurück
- *public int getQoS()* gibt das aktuelle QoS zurück
- *public int getServiceProviderQos()* gibt das aktuelle QoS des Service Providers zurück
- *public int getNewQos()* gibt das geänderte QoS zurück
- *public int getNewServiceProviderQoS()* gibt das geänderte QoS des Service Providers zurück
- *public CosimUsedPolicy getPolicy()* gibt aktuelle Policy zurück
- *public CosimUsedPolicy getServiceProviderPolicy()* gibt die Policy des Service Providers zurück

3.4.1.3 Die Entity-Klasse CosimUsedPolicy

Abbildung der Tabelle SIMULATION_USED_POLICIES

Attribute

- *private int simulationId* Id der aktuellen Simulation
- *private int roundId* Nummer der aktuellen Runde
- *private int peerId* aktuelle Peer Id
- *private int serviceProviderId* aktuelle Service Provider Id
- *private int policyId* aktuelle Policy Id
- *private int serviceProviderPolicy* aktuelle Policy des Service Providers

Methoden

- *public int getRoundId()* gibt die jeweilige Rundennummer zurück
- *public CosimMember getPeer()* gibt die jeweilige Peer zurück
- *public CosimMember getServiceProvider()* gibt die jeweilige Service Provider zurück
- *public CosimUsedPolicy getPolicy()* gibt die Policy zurück
- *public CosimUsedPolicy getServiceProviderPolicy()* gibt die Policy des Service Providers zurück

3.4.1.4 Die Entity-Klasse CosimFeedback

Abbildung der Tabelle SIMULATION_FEEDBACK

Attribute

- *private int simulationId*
- *private int roundId*
- *private int contextId*
- *private int faceId*
- *private int peerId*
- *private int serviceProviderId*
- *private int outcome*

Methoden

- *public int getRoundId()* gibt die Rundennummer aus
- *public int getContextId()* gibt die Context Id aus
- *public int getFaceId()* gibt die Face Id aus
- *public CosimMember getPeer()* gibt den Peer zurück
- *public CosimMember getServiceProvider()* gibt den Service Provider zurück
- *public int getOutcome()* gibt den Outcome aus

3.4.1.5 Die Entity-Klassse CosimSimulation

Abbildung der Tabelle SIMULATIONS

Attribute

- p*rivate int simulationId* Ist die Id der aktuellen Simulation
- *private String description* Spielbeschreibung
- *private int currentRound* In welcher Runde sich das Spiel befindet
- *private int totalRounds* Maximale Anzahl der Spielrunden
- *private int currentStatus* Der aktuelle Status
- *private Date creationDate* Das Erstellungsdatum
- *private Date startTime* Der Simulationsstart

Methoden

- *public int getSimulationId()* Gibt die Id der instanzierten Simulation aus
- *public String getDescription()* Gibt die Beschreibung der aktuellen Simulation aus
- *public int getCurrentRound()* Gibt die aktuelle Rundennummer aus
- *public int getTotalRounds()* Gibt die maximale Rundenummer aus
- *public int getCurrentStatus()* Gibt den aktuellen Status aus
- *public Date getCreationDate()* Gibt das Erstellungsdatum aus
- *public Date getStartTime()* Gibt das Datum des Simulationsstartes aus

3.4.2 Datenstrukturen (server.model.datastructures)

Hier befinden sich Datenstrukturen, die das server Package benutzt. Die zentrale Datenstruktur ist hier Graph.

3.4.2.1 Die Klasse GraphModel

Diese Klasse stellt die Verwaltung des Supergraphen. Sie übernimmt den Abgleich des persistent gehaltenen Supergraphen mit der Datenbank und versorgt denGraphCreator mit Graph-Instanzen. Zudem wird von java.util.Observable geerbt, um von CachedGraphRequests beobachtet zu werden und diese bei Änderungen zu informieren. So können sich CachedGraphRequests löschen, sobald sie veraltet sind.

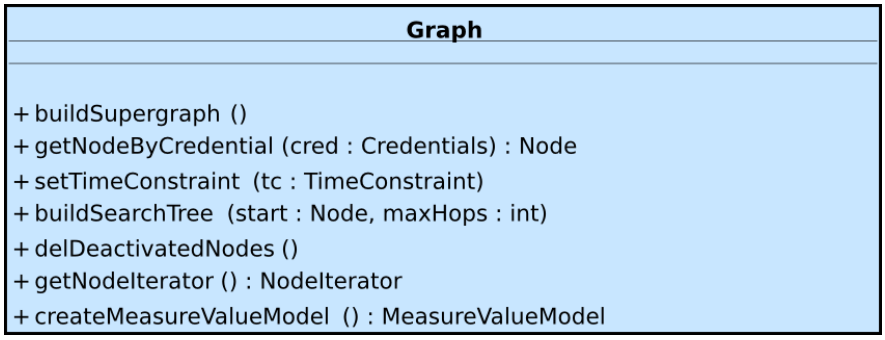
Attribute

- *private graph Graph* Der Supergraph

Methoden

- *public Graph getGraph* Gibt den gespeicherten Graphen zurück.

3.4.2.2 Die abstrakte Klasse Graph

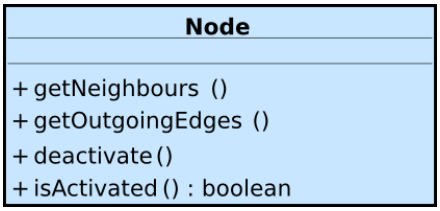


Die Datenstruktur repräsentiert einen Hypergraphen. Diese Klasse ist abstrakt, da Unterklassen eine spezielle Graphrepräsentation implementieren.

Methoden

- *public void buildSupergraph()* Baut den Supergraphen aus den Datenbankklassen. Der Supergraph ist die Vereinigung aller Graphen der Spielrunden in Cosim. Zu jedem Knoten wird gespeichert, in welchen Runden er existiert. BeiGraphanfragen wird dieser Graph geklont und auf die Anfrage zurechtgeschnitten.
- *public Node getNodeByCredential(Credential cred)* Liefert einen Node zu einem Credentials, nützlich um den Knoten des eingeloggten Users zu erfragen.
- *public void setTimeConstraint(TimeConstraint tc)* Setzt eine zeitliche Begrenzung auf den Graphen. Alle Knoten des Graphen, die nicht in diesem Zeitfenster existieren, werden entfernt.
- *public void buildSearchTree(Node start, int maxHops)* Baut einen Suchbaum ausgehend vom Starknoten start auf mit maximaler Höhe maxHops.
- *public void delDeactivedNodes()* Löscht alle deaktvierten Knoten des Graphen.
- *public void getNodeIterator()* Gibt einen Iterator für die Knotenmenge zurück.
- *public MeasureValueModel createMeasureValueModel()* Diese Fabrikmethode erzeugt ein neues MeasureValueModel und speichert ein Verweis zu diesem Model.

3.4.2.3 Die abstrakte Klasse Node



Durch diese Klasse werden Knoten im Graph modelliert. Knotengewichte werden mit MeasureValueModel modelliert.

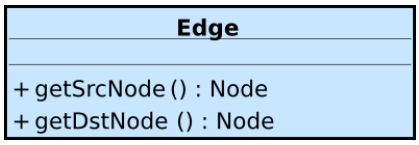
Attribute

- *private boolean active* True wenn der Knoten aktiviert ist (true bei Erstellung).

Methoden

- *public Set<Node> getNeighbours()* Gibt die benachbarten Knoten.
- *public Set<Edge> geOutcomingEdges()* Gibt ausgehende Kanten.
- *public void deactivate()* Deaktiviert den Knoten
- *public booleacn isActivate()* Gibt true zurück, wenn der Knoten aktiviert ist

3.4.2.4 *Die abstrakte Klasse Edge*

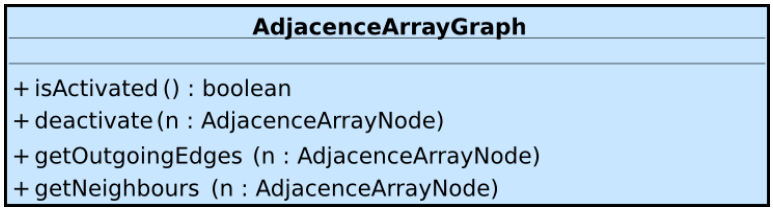


Eine Instanz dieser Klasse repäsentiert eine Kante im Graphen.

Methoden

- *public Node getSrcNode()* Gibt den Ausgangsknoten dieser Kante
- *public Node getDstNode()* Gibt den Zielknoten dieser Kante

3.4.2.5 *Die Klasse AdjacenceArrayGraph*

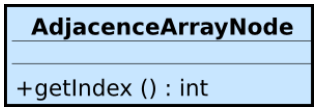


Hierbei handelt es sich um die Adjazenzarrayimplementierung der Graphendatenstruktur Graph. Modelliert werden Kanten und Knoten dieses Graphen durch AdjacenceArrayEdge und AdjacenceArrayNode.

Methoden

- *public boolean isActivated(AdjacenceArrayNode n)* Gibt zurück, ob der gegebene Knoten aktiviert ist oder nicht.
- *public void deactivate(AdjacenceArrayNode n)* Deaktiviert den gegebenen Knoten im Graphen
- *public Set<Edge> getOutgoingEdges(AdjacenceArrayNode n)* Gibt die ausgehenden Kanten eines Knoten zurück.
- *public getNeighbours(AdjacenceArrayNode n)* Gibt die benachbarten Knoten des gegebenen Knoten.

3.4.2.6 *Die Klasse AdjacenceArrayNode*



Implementiert Node für die Graphenrepräsentation als Adjazenzarray.

Attribute

- *private int index* Position des Knotens im Adjazenzarray

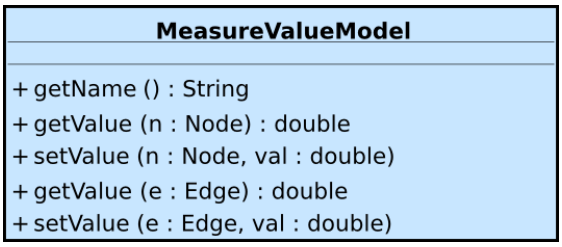
Methoden

- *public int getIndex()* Gibt Index zurück

3.4.2.7 Die Klasse AdjacenceArrayEdge

Implementiert Edge für die Graphenrepräsentation Adjazenzarray.

3.4.2.8 Die Klasse MeasureValueModel



Mithilfe dieser Klasse fügen Plugins Kanten- und Knotengewichte dem Graphen hinzu. Alle Instanzen werden durch die Fabrikmethode Graph.createMeasureValueModel() erstellt und sind diesem Graphen zugehörig.

Attribute

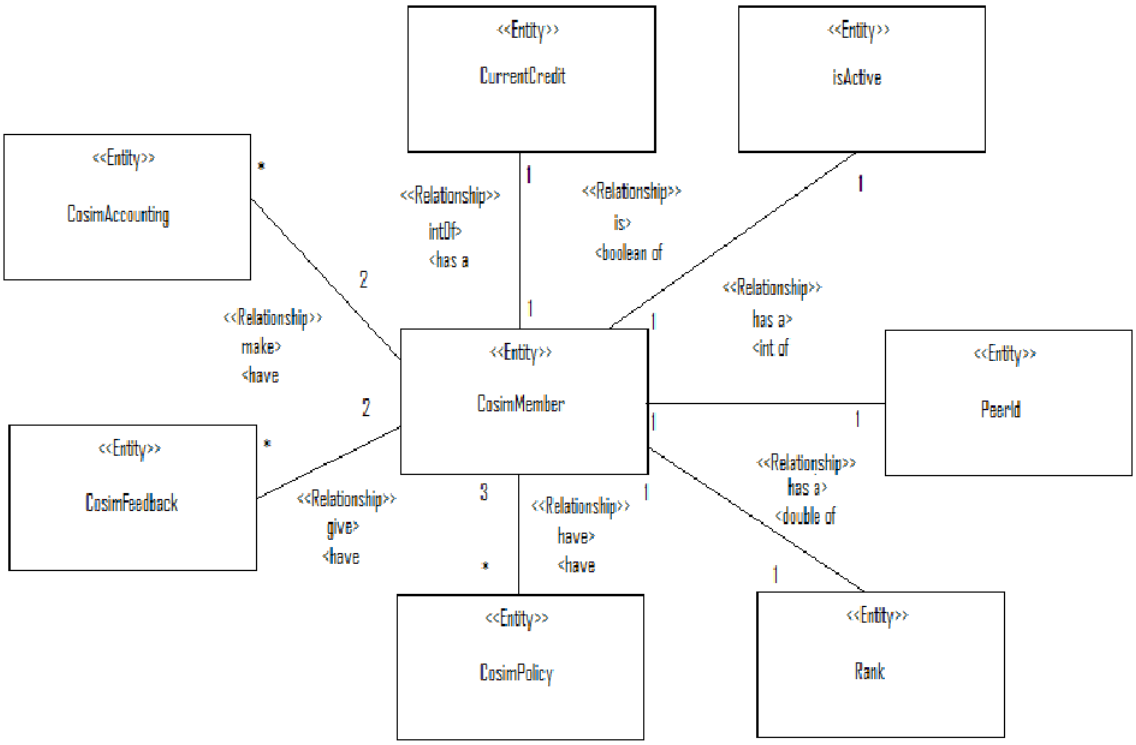
- *private String name* Name des Models, üblicherweise gleichzeitig des Zentralitätsmaßes

Methoden

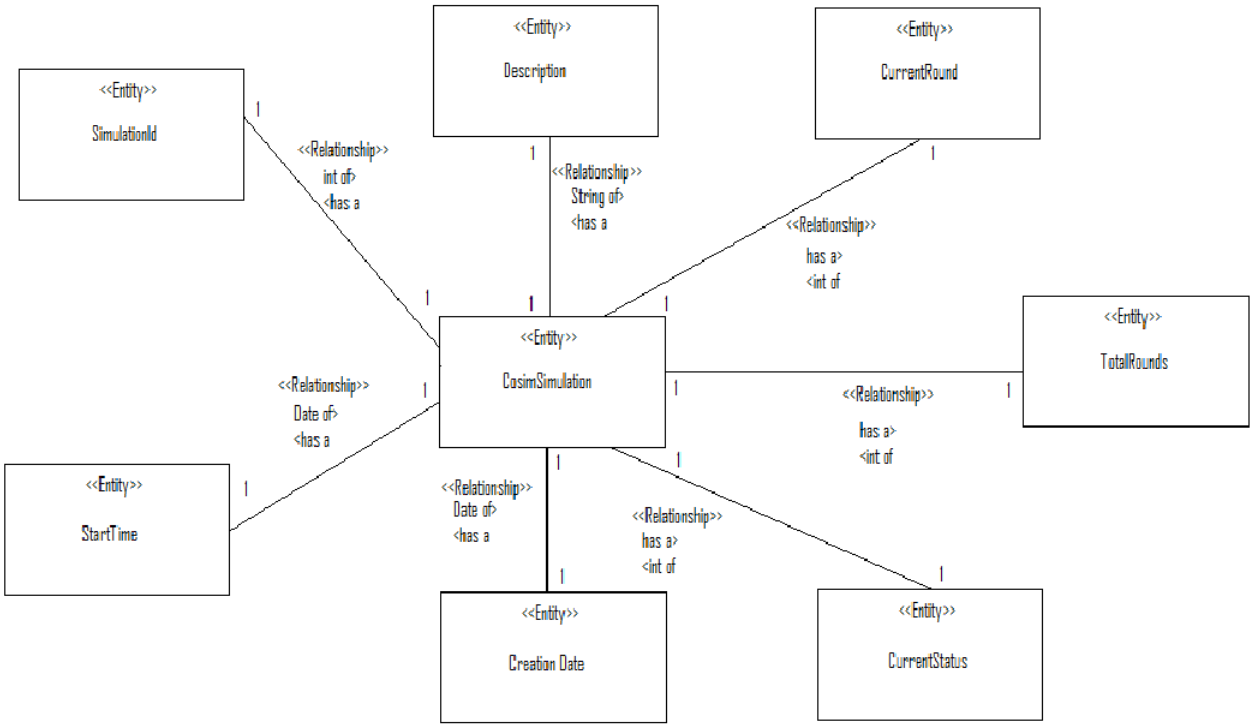
- *public String getName()* Gibt den Namen des Models zurück.
- *public double getValue(Node n)* Gibt das zu dem gegebenen Knoten gespeicherte Knotengewicht zurück
- *public void setValue(Node n, double val)* Setzt das Knotengewicht des Knotens.
- *public double getValue(Edge e)* Gibt das zu der gegebenen Kante gespeicherte Kantengewicht zurück
- *public setValue(Edge e, double val)* Setzt das Kantengewicht einer Kante

4 Beziehungen (entity-relations)

4.1 CosimMember

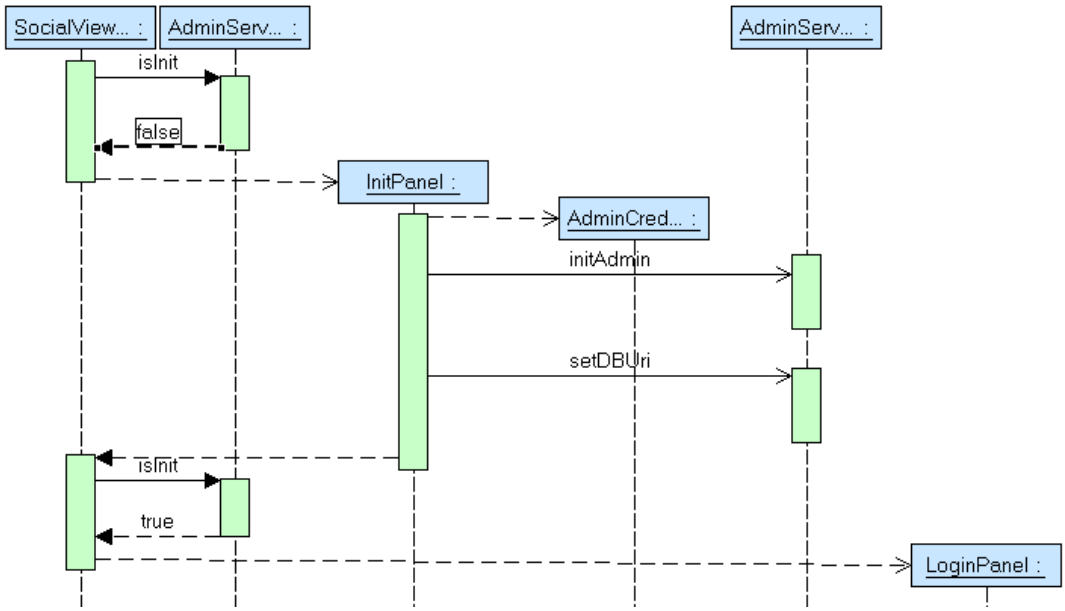


4.2 CosimSimulation



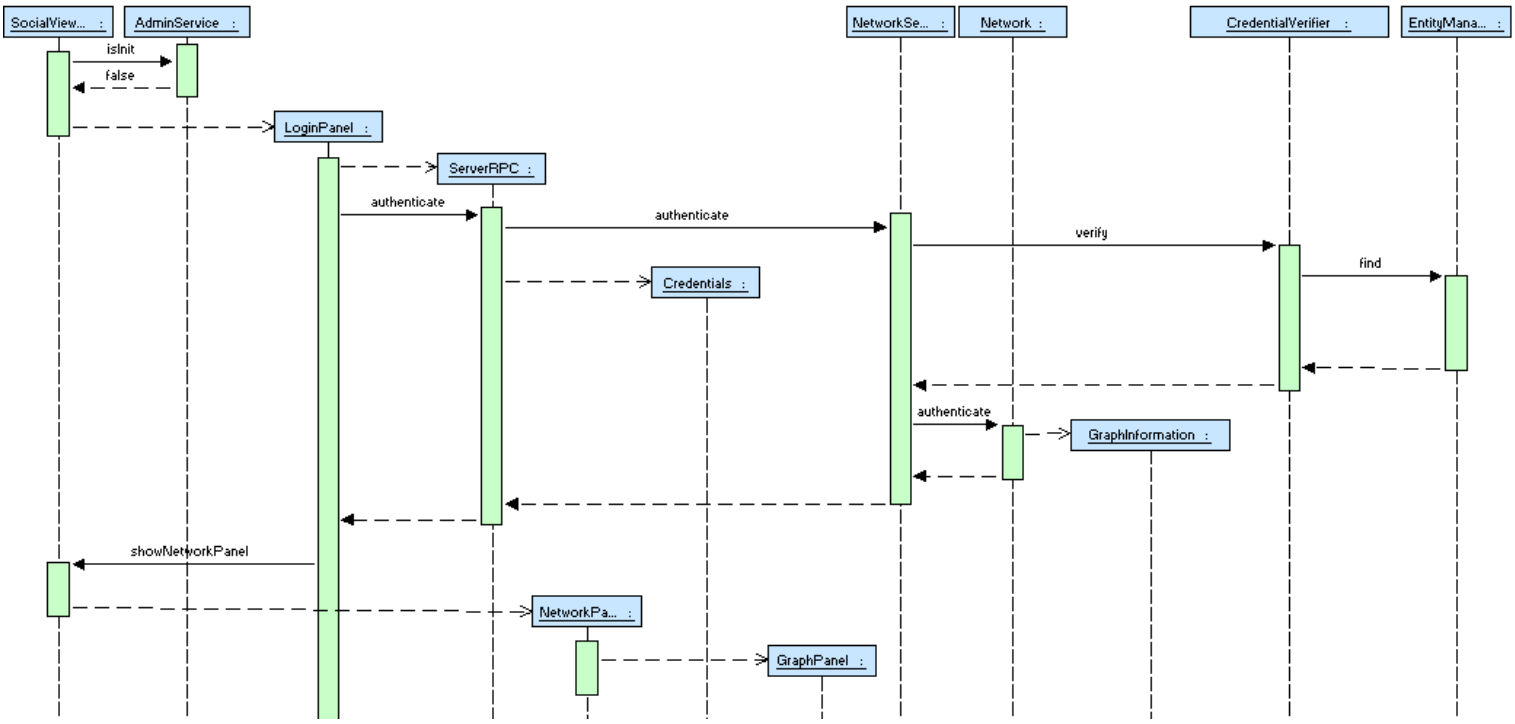
5 Abläufe

5.1 Initalisierung



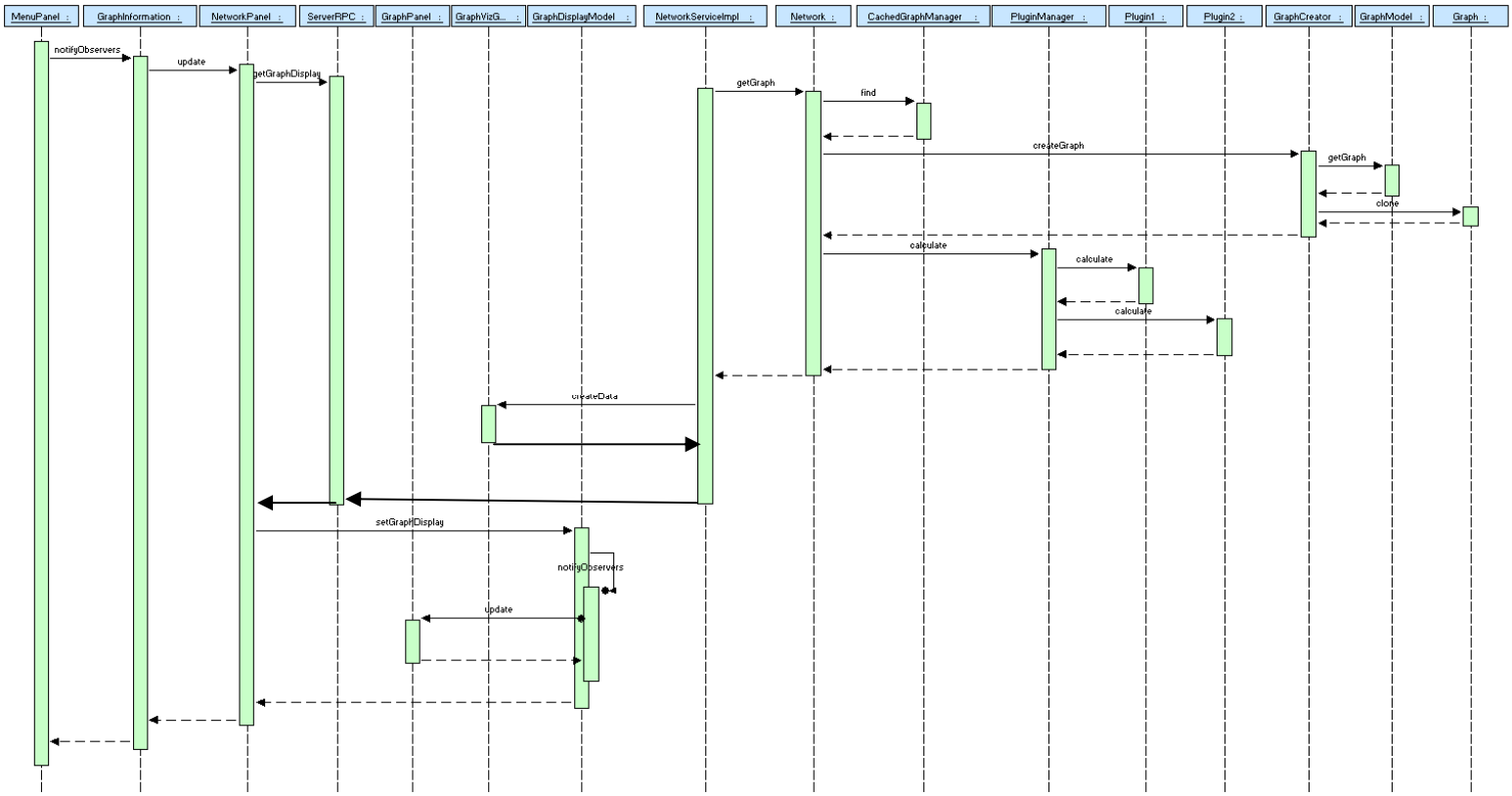
SocialViewEntryPoint fragt bei AdminService über die Funktion isInit() an, ob das Spiel bereits initialisiert wurde. Falls dies nicht der Fall ist, initialisiert der EntryPoint eine Instanz von InitPanel. Hier gibt der User die Credentials für den Admin und die Datenbankverbindungsdaten ein. Nach dem Submit schickt der Client über initAdmin() in der AdminService Schnittstelle die neuen Admin Credentials an den Server. Diese Daten speichert der Server in der persistenten Datenbank. Nachdem dieser Aufruf abgeschlossen wurde, werden die Datenbankverbindungsdaten über die Schnittstelle setDBUri an den Server übertragen. Diese Daten werden ebenfalls in der Datenbank gespeichert. Danach wird InitPanel beendet und SocialViewEntryPoint ruft dasLoginPanel auf.

5.2 Login



Die Interaktion beginnt beim Aufruf der Webseite mit dem Initialisieren des SocialViewEntryPoints. Da noch keine Netzwerk vorhanden ist wird das LoginPanel initialisiert, in dem der User seine Credentials eingibt. Nach dem Abschicken der Daten wird aus den Login-Daten ein neues Credentials-Objekt gebaut und an die authenticate Schnittstelle vonNetworkServices geschickt. Auf der Server Seite überprüft NetworkServicesImpl mittels CredentialVerifier und dessen Methode verifiy(), ob das übergebene Credentials-Objekt einem User zugeordnet ist. CredentialVerifier überprüft mittels des EntityManagers der JPA in der Datenbank ob dies zutrifft. Ist dies der Fall fragt NetworkServiceImpl bei Network an, um ein NetworkGraph-Objekt zu bekommen, welchen er wieder zurück an den LoginPanel schickt. Falls bei LoginPanel ein NetworkGraph-Objekt ankommt, ist der Login Prozess beendet und LoginPanel überreicht das NetworkGraph-Objekt zurück an SocialViewEntryPoint welcher nun das LoginPanel schließt und ein NetworkPanel mit dem NetworkGraph-Objekt initialisiert und anzeigt.

5.3 Graph anfordern



Nach der Auswahl der gewünschten Optionen zur Erstellung des Graphen benutzt der Client den Absenden Button. Das veranlaßt das MenuPanel den NetworkGraph mit den geänderten Optionen zu updaten. Sobald er fertig ist, ruft er im NetworkGraph notifiyObservers() auf was das NetworkPanel auf die Änderungen aufmerksam macht. Dies veranläßt das NetworkPanel einen neuen Graphen bei NetworkService anzufordern. Diese Anfrage delegiert auf der Serverseite NetworkServiceImpl an Network. Network fragt eventuell eim CachedGraphModel nach, ob es diesen Graphen bereits in berechneter Form gibt. Ist dies nicht der Fall wird in GraphFactory die Erstellung eines neuen Graphen gestartet. In GraphFactory wird von GraphModel erst der komplette Graph aus JPA angefordert und an Network zurückgeschickt. Hier wird dieser Graph an den Pluginmanger übergeben. Hier wird der Graph, über calculate(), von Plugin zu Plugin weitergereicht und am Ende der Berechnung wieder an Network zurückgeben. Network erstellt nun einGraphVizGraphDisplay-Objekt, was zurück an die Client Klasse GraphPanel, welcher den GraphVizModel updatet und es zurück an NetworkPanel schickt, welcher GraphPanel updatet.