KIT – Karlsruhe Institut of Technology

Praxis der Software-Entwicklung WS 2009/10

Gruppe 7

Karlsruhe Institute of Technology

»Implementierungsheft« v 1.0

Analysesoftware für Soziale Netzwerke

10.02.2010

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 3
2 Probleme und Änderungen am Entwurf 4
2.1 Ursprüngliches Klassendiagramm 4
2.2 Änderungen im Package client.gui 5
2.2.1 Die Klasse SocialViewEntryPoint 5
2.2.2 Die (ehemalige) Klasse MenuPanel 5
2.2.3 Die neue Klasse NetworkTopPanel 7
2.2.4 Die Klasse ServerRpc 7
2.2.5 Weitere Hilfsklassen 7
2.3 Änderungen im Package server.controller 8
2.3.1 Das neue Package server.controller.graphviz 8
2.3.2 Änderungen im Package server.controller.persistant 10
2.4 Änderungen im Package server.model 11
2.4.1 Änderungen im Package server.model.cosim 11
2.4.2 Änderungen im Package server.model.datatstructures 12
3 Implementierungsplan 15

1 Einleitung

Dieses Abschlussdokument der Implementierungsphase dient als Beschreibung dieser Phase und zeigt insbesondere Probleme bei der Implementierung und daraus resultierende Änderungen im Vergleich zum Entwurf auf.

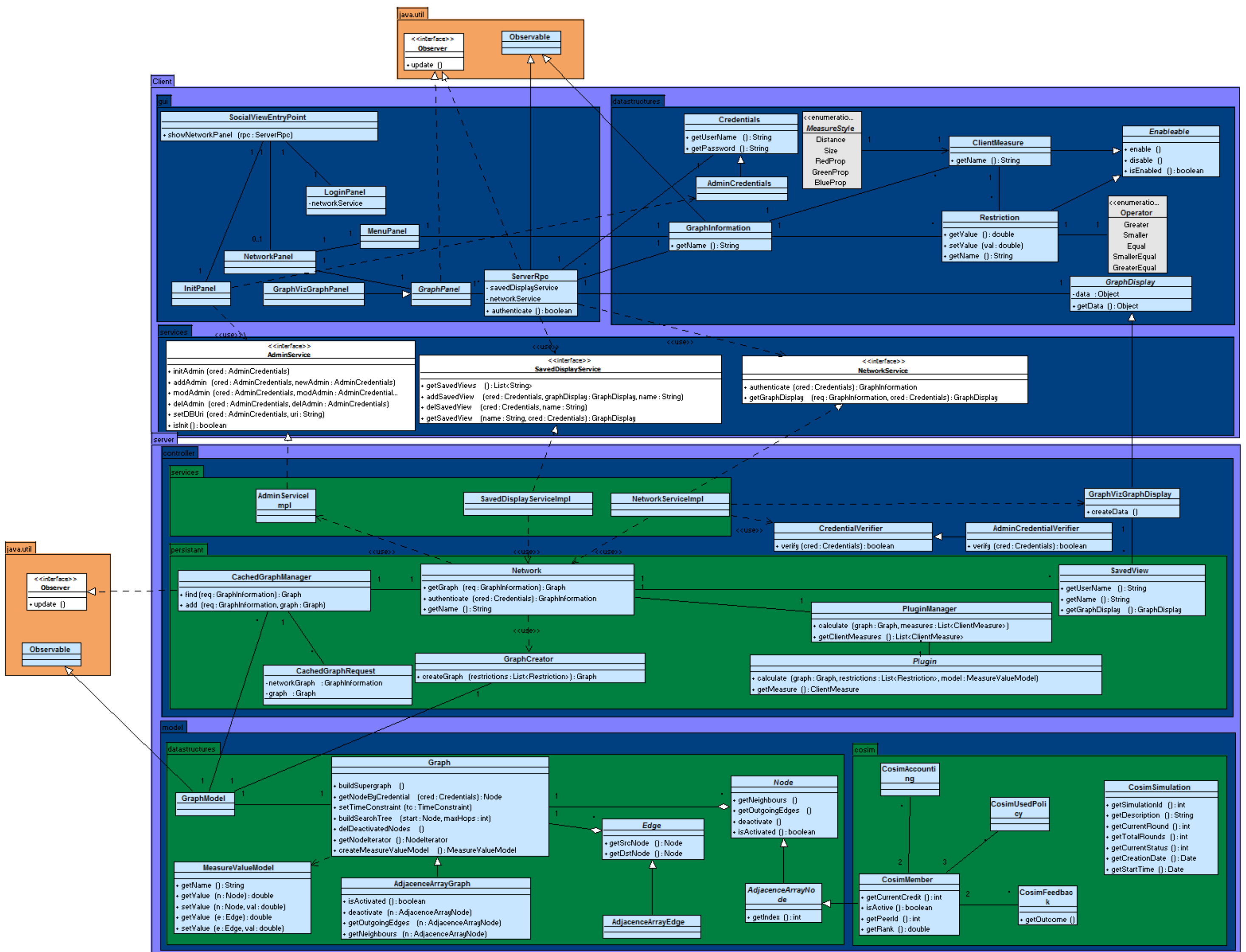
Die Implementierungsphase wurde mit einem sehr straffen Zeitplan durchgeführt, der jedoch trotzdem grob eingehalten werden konnte. Dies kann darauf zurückgeführt werden, dass trotz zahlreichen kleinen Änderungen an einigen Schnittstellen die grobe Struktur und Interaktionslogik zwischen einzelnen Klassen und Paketen nicht überarbeitet werden musste.

Die tiefgreifenderen Änderungen am Entwurf sind hauptsächlich Folge von Implementierungsdetails, die die Graphenvisualisierungsbibliothek GraphViz, die Datenbankschnittstelle JPA oder GWT betreffen. Außerdem wurde der Entwurf des Userinterfaces verändert, um die Trennung von Darstellung und Logik strenger durchsetzen zu können. Da jedoch keine Package-Schnittstellen verändert wurden, blieben diese Änderungen ohne weitreichende Auswirkungen.

2 Probleme und Änderungen am Entwurf

In diesem Kapitel werden die Änderungen, die sich während der Implementierung ergeben haben dokumentiert. Die Pakete werden in logischer Reihenfolge in einem Top-Down Ansatz behandelt. Anmerkung: Packagenamen werden ohne dem Präfix *edu.kit.pse.socivalview* notiert. Pakete und Klassen, die unberührt blieben, werden nicht erwähnt.

2.1 Ursprüngliches Klassendiagramm



2.2 Änderungen im Package client.gui

2.2.1 Die Klasse SocialViewEntryPoint

Die Klasse selbst wurde nicht erweitert. Intern wurde ein State Pattern benutzt um zu entscheiden welches Panel geladen wird.

2.2.2 Die (ehemalige) Klasse MenuPanel

Eine Implementierung des Menüs in einer Klasse hätte in einer viel zu großen Klasse resultiert. Deshalb und aus Gründen der Wiederverwendbarkeit (Für globale Restrictions, sowie für jedes Measure kann das gleiche RestrictionPanel verwendet werden) wurde die Klasse in folgende logische Unterklassen eingeteilt

2.2.2.1 Die (geänderte) Klasse MenuPanel

Die momentane Implementation von MenuPanel enthält folgende Funktionalität.

Methoden

- *void addWidget(Widget w)* Fügt ein Widget in das MenuPanel ein.
- *void removeWidget(Widget w)* Entfernt folgendes Widget aus dem MenuPanel
- *GraphInformation getGraphInformation()* Gibt ein aus den Einstellungen des Menus erstellten GraphInformation-Objekt zurück. Logik zur Erstellung wird im Listener implementiert.
- *void reenable()* Reaktiviert den Submit-Button des MenuPanels.

2.2.2.2 Das Interface MenuListener

Das MenuListener Interface ist im MenuPanel als statisches Interface definiert. Eine Implementierung dieses Interfaces kann als Logik für das Menu benutzt werden.

Methoden

- *void onInit(MenuPanel p)* Operationen die die Logik beim Initialisieren des Panels aufrufen möchte, werden in dieser Funktion definiert
- *void onSubmit(MenuPanel p)* Enthält die Logik für Klicken des Buttons
- *GraphInformation getGraphInformation(MenuPanel p)* Enthält Logik um das GraphInformation-Objekt aus dem Menu zu erstellen.

2.2.2.3 Die Klasse RestrictionPanelContainer

Enthält eine Liste von aktiven Restriktionen zur Anzeige.

Methoden

- *void addWidget(Widget w)* Fügt ein Widget in den Container ein
- *List<Restriction> getRestrictions()* Gibt die Liste aller aktiven Restriktionen aus. Funktionalität befindet sich in der Logik
- *void removeWidget(Widget w)* Entfernt Widget w
- *void reenable()* Reaktiviert den Submit-Button

2.2.2.4 Die Klasse MeasurePanelContainer

Diese Klasse ist das Analogon zu RestrictionPanelContainer für Measures.

2.2.2.5 Die Klasse RestrictionPanel

Diese Klasse visualisiert eine Restriktion im Userinterface.

Methoden

- *Operator getOperator()* gibt aktuellen Operator zurück
- *Restriction getRestriction()* gibt aktuelle Restriktion zurück (implementiert in Logik)
- *getRestrictionName()* gibt Namen der Restriktion zurück
- *double getValue()* gibt aktuelles Value zurück
- *void setOperator(Operator)* setzt den aktuellen Operator
- *void setRestrictions(List<Restriction>)* setzt aktuell verfügbare Restriktionen
- *void setValue(double)* setzt das aktuelle Value

2.2.2.6 Die Klasse MeasurePanel

Diese Klasse visualisiert (analog zu RestrictionPanel) ein Zentralitätsmaß im Userinterface.

Methoden

- *void disableRestrictions()* entfernt die zum ClientMeasure gehörenden Restriktionen
- *ClientMeasure getMeasure()* gibt das aktuell ausgewählte Measure zurück
- *String getSelectedMeasureString()* gibt den aktuellen ausgewählten Namen des Measures zurück
- *String getSelectedStyleString()* gibt den Namen des aktiven Styles zurück
- *void setActiveMeasure(ClientMeasure)* setzt das aktuelle Measure
- *void setAvailableMeasures(List<ClientMeasure>)* setzt die verfügbaren Measures
- *void setAvailableMeasureStyles(List<MeasureStyle>)* setzt die verfügbaren Styles
- *void setMeasureStyle(MeasureStyle)* setzt das aktuelle Style
- *void setRestrictionListener(RestrictionContainerListener)* setzt die Logik für das jeweilige RestrictionContainerPanel im MeasurePanel

2.2.2.7 Die Klasse PeerPanel

Dieses Panel enthält eine Liste aller Knoten und entscheidet welches zentral angezeigt wird. Da dieser Klasse eine Liste von Strings übergeben werden und das vom User selektierte String zurückgegeben wird, enthält diese Klasse keine Logik und deshalb auch keinen Listener.

Methoden

- *String getSelected()* gibt ausgewählten String zurück
- *void setItems(List<String>)* setzt die verfügabren Strings

2.2.3 Die neue Klasse NetworkTopPanel

Enthält die obere Menu Bar mit Namen des Netzwerkes, Username und dem Logout Button. Dies sollte ursprünglich von NetworkPanel übernommen werden.

Methoden

- *String getNetwork()* gibt das ausgewählte Netzwerk aus
- *String getUsername()* gibt den aktiven User aus
- *void setUsername(String)* setzt den aktuellen User
- *void setNetwork(String)* setzt das aktuelle Netzwerk

2.2.4 Die Klasse ServerRpc

Eine unserer Forderungen an das UI ist, dass die Logik von der Benutzeroberfläche getrennt ist. In unserem Entwurf wollten wir dies über die Klasse ServerRPC erreichen. Da die Klasse, wie MenuPanel, viel zu groß geworden wäre und zu viele Abhängigkeiten hätte, entschieden wir uns die Logik für jedes Panel zu entkoppeln. Deswegen definiert jedes Panel ein Interface über das eine Logik für die Panel definiert werden kann um das jeweilige Panel steuert. Die ServerRPC Klasse wurde für jedes Panel entkoppelt und in das Package client.logic abstrahiert. Die Einstiegsklasse für unsere Implementation der UI-Logik ist die Klasse NetworkLogic. Von hier aus werden, analog zur Panel-Struktur, weitere Logic-Instanzen eingebunden.

2.2.5 Weitere Hilfsklassen

2.2.5.1 Services

Diese statische Klasse erleichtert den Zugriff auf die asynchronen Service Interfaces. Da durch diese Klasse der Initialisierungscode für Services innerhalb des Programms wegfällt, haben wir uns entschieden diese Klasse zu verwenden. Die Klasse ähnelt dem Entwurfsmuster Singleton

Methoden

- *AdminServiceAsync getAdminServiceInstance()* gibt Admin Servlet zurück
- *NetworkServiceAsync getNetworkServiceInstance()* gibt Network Servlet zurück

2.2.5.2 IconBarPanel

Dieses Panel wird von MeasurePanel und RestrictionPanel benutzt, um einen Rahmen mit einem Close-Button um die Panels zu zeichnen.

Methoden

- *void setTitle(String)* Setzt den Namen des Panels

Das Interface StateListener implementiert eine onClose(IconBarPanel) Funktion

2.2.5.3 MessagePanel

Mit diesem Panel können Nachrichten für den User im UI angezeigt werden. Vereinfacht den Zugriff auf GWTs PopupPanel.

2.2.5.4 LogicPanelMapper

Ist eine Hashmap die von den Container Logiken benutzt wird um Panel Logik auf Panel zu mappen. Somit kann der Container ein Panel entfernen, wenn die entsprechende Panel Logik dies veranlasst.

Methoden

- *void add(K, V)* Fügt ein Tupel hinzu
- *Set<K> getListener()* Gibt die Listener aus
- *Collection<V> getPanels()* Gibt die Panels aus
- *V removeByListener(K)* entfernt ein Panel über den korrespondierenden Listener

2.3 Änderungen im Package server.controller

2.3.1 Das neue Package server.controller.graphviz

Das Package enthält nach dem "Prinzip des antizipierten Wandels" die Logik um ein *GraphDisplay* im Sinne von GraphViz zu implementieren. Die Klassen des Packages realisieren das GOF-Entwurfsmuster *Abstact Factory*.

Das vorliegende Package dient dazu, GV Source zu erstellen, der dann von GraphViz weiter verarbeitet werden kann. Damit ersetzt es die Logik, die im Entwurf in *GraphVizGraphDisplay* gekapselt war. Die Entkopplung dessen wurde mit der zunehmender Einsicht der Komplexität dieser Aufgabe unausweichlich.

2.3.1.1 Die Klasse GraphVizGraphDisplayCreator (gof - abstract factory - client)

Methoden

- *GraphDisplay createGraphDisplay(Graph graph, GraphInformation req)* Diese Methode erstellt aus der fertig berechneten Graph-Datenstruktur und einer GraphInformation-Instanz, die Informationen zu der Darstellung der berechneten Zentralitätsmaße enthält, ein GraphVizGraphDisplay. Dieses Objekt kapselt einen String in XDOT-Syntax. Zur Erstellung dieses Quellcodes werden das von GraphViz gelieferte Dot-Programm und die jeweiligen Factories weiter unten benutzt.

2.3.1.2 Die Klasse DotFactory (gof - abstract factory - abstract factory)

Diese abstrakte Klasse stellt die Oberklasse für die konkreten implementierenden Factories zur Verfügung. Sie definiert abstrakte Methoden zur Realisierung der createMethode im Factory-Pattern.

Methoden

- *void setSizes(DotElement[] elements, MeasureValueModel mVM)* die Methode dient dazu Bei den Elementen, die die Factories kreieren das Attribut Size zu Setzen
- *void setDistances(DotElement[] elements, MeasureValueModel mVM)* die Methode dient dazu Bei den Elementen, die die Factories kreieren das Attribut Distance zu Setzen
- *abstract double[] getValues(DotElement[] elements, MeasureValueModel mVM)* die Methode dient dazu Bei den Elementen, die die Factories kreieren das Attribut Value zu Setzen
- *abstract DotElement[] createDotElements()* Die Methode ist die create - Methode der abstract Factory. Sie erschafft also Instanzen des abstrakten Produkts *DotElement.*

2.3.1.3 Die Klassen DotNodeFactory und DotEdgeFactory (gof - abstract factory - concrete factory)

Diese Klassen implementieren DotFactory und erstellt DotNode beziehungsweise DotEdge Instanzen.

2.3.1.4 Klasse DotElement (gof - abstract factory - abstract product)

Klasse zur Realisierung des abstrakten Produkts innerhalb des Factory-Patterns. Die abstrakten Methoden dienen dazu, die Syntax (also konkret Knoten oder Kante) zu ermitteln, das auf das konkrete Element abbildet.

Methoden

- *void setColor(String colorName, double color)* die Methode dient dazu das Attribute Color zu setzen
- *void setSize(double size)* die Methode dient dazu das Attrbut Size zu setzen
- *String generateDotString()* die Methode dient dazu die Stringrepräsentation des DotElement zu erzeugen
- *abstract String generateElementName()* die Methode kreiert den String zur Namensrepräsentation des Elements
- *String generateElementColor()* die Methode kreiert den String zur Farbrepräsentation des Elements
- *String generateElementFontColor()* die Methode setzt den String zur Schriftfarbe wenn nötig
- *abstract String generateElementSize()* die Methode kreiert den String zur Größenrepräsentation des Elements
- *abstract String generateElementDist()* die Methode kreiert den String zur Distanzrepräsentation des Elements
- *GraphElement getElement()* die Methode gibt das Element

2.3.1.5 Die Klassen DotNode und DotEdge (gof - abstract factory - concrete product)

Diese Klasse implemetieren DotElement für die konkreten Produkte Knoten und Kante.

2.3.2 Änderungen im Package server.controller.persistant

Die Namensgebung dieses Packages war ungünstig, deswegen wurde es im Nachhinein in *server.controller.kern* umbenannt, um die Funktion der Klassen dieses Packages klarer herauszustellen.

2.3.2.1 Die Klasse SavedDisplayServiceImpl

Diese Klasse diente zur Implementierung von Wunschkriterien, die nicht mehr erfüllt werden konnten. Dadurch wurde diese Klasse überflüssig.

2.3.2.2 Die neue Klasse PackageAnalyzer

Diese Utility-Class von http://www.tutorials.de/forum/java/252078-packetnamen-aus-einem-packet-auslesen.html liest Paket- und Klassennamen aus. Sie findet ihre Verwendung bei dem automatischen Laden von Plugins. Um Kompatibilität zwischen verschiedenen Betriebssystemen zu gewährleisten, konnte diese Klasse nicht direkt übernommen werden, sondern holt nun die Trennzeichen im Classpath aus einer Systemvariablen.

Methoden

- *static List<String> findAllClassesContainedBy(Package thePackage)* Gibt eine Liste aller Klassennamen in einem Package zurück.

2.3.2.3 Die Klassen CachedGraphManager und CachedGraphRequest

Caching gehörte auch zu den Wunschkriterien aus dem Pflichtenheft. Caching sollte zum Einen durch die Persistierung des Supergraphen, zum Anderen durch das Cachen von spezifischen Graphenanfragen durch diese beiden Klassen realisiert werden. Nach der Implementierung von Persistierungsmechanismen des Supergraphen konnten bereits keine großen Performanceprobleme mehr festgestellt werden, sodass zunächst auf das erweiterte Caching hier verzichtet werden konnte. Sollten sich im Zuge von Stresstests in der Testphase doch Performanceprobleme ergeben wird hier die Implementierung nachgereicht.

2.3.2.4 Die Klasse SavedView

Diese Klasse diente zur Implementierung von Wunschkriterien, die nicht mehr erfüllt werden konnten. Dadurch wurde diese Klasse überflüssig.

2.3.2.5 Die Klasse GraphCreator

Methoden

- *getInstance()* Rückgabe des Singletons
- *getRestrictions()* Gibt eine Liste von den vom GraphCreator implementierten Restrictions zurück. Somit sind nur noch an einer Stelle im Code Änderungen nötig, wenn neue globale Restrictions implementiert werden.

2.3.2.6 Die Klasse Network

Diese Klasse wird mittels nun mittels einer Credentials Instanz instanziert, das einige Methodenschnittstellen verändert. Grund für diese Designentscheidung war, dass eine Credential-Instanz in verschiedenen Methoden benötigt wird und eine Instanzierung von Network nur mit den gültigen Credentials eines Users Sinn ergibt.

Methoden

- *authenticate(credentials)* Diese Methode wurde umbenannt in *getGraphInformation()*, da die Verifizierung des Benutzers in NetworkServiceImpl verschoben wurde.
- *List<String> getUserList()* gibt eine Liste von Usernamen zurück, damit einem Administrator die Auswahl zwischen verschiedenen Peersichten im Userinterface angezeigt werden kann.

2.3.2.7 Die neue Klasse CalculateRunner

Der Aufruf von calculate() der Plugins im PluginManager wurde durch Threads realisiert, um diese Berechnungen parallel laufen lassen zu können. Aus technischen Gründen wurde diese Klasse hinzugefügt, die einen ein Zentralitätsmaß berechnenden Thread repräsentiert.

2.4 Änderungen im Package server.model

2.4.1 Änderungen im Package server.model.cosim

2.4.1.1 Die neue Klasse CosimEntityManager

Diese Klasse stellt einen Singleton EntityManager für die Cosim Datenbank zur Verfügung. Die Notwendigkeit dafür konnte in der Entwurfsphase ohne Kentniss von der genauen Funktionsweise des Entity Managements in JPA nicht gesehen werden.

2.4.1.2 Die neuen JPA Primary-Key Klassen

In der Cosim-Datenbank besitzen einige Views zusammengesetzte Primary-Keys. Sobald der Primary-Key einer auf eine Entity Bean Class gemappten Tabelle nicht mehr nur aus einem Feld besteht, müssen in JPA sogenannte Primary-Key Klassen verwendet werden, um korrekt abbilden zu können. Dies war zur Entwurfsphase noch nicht bekannt. Aus diesem Grund kamen die Klassen *CosimMemberKey*, *CosimFeedbackKey*, *CosimAccountingKey* und *CosimIntervalKey* neu dazu. All diese Klassen haben speziell auf die konkreten Primary Keys angepasste Konstruktoren und Getter für die einzelnen Felder. Außerdem wird jewails eine eigene Implementierung von equals mitgeliefert. Die Struktur der Klassen werden fast ausschließlich durch JPA und dem Cosim Datenbankschema vorgegeben.

2.4.1.3 Die Klasse CosimMember

Die Vererbungsstruktur zwischen CosimMember und AdjacenceArrayNode wurde aufgegeben und durch Getter auf beiden Seiten ersetzt, die die eine normale

Assoziation implementieren. Die Vererbungshirachie ist sehr ungünstig, da zwar einige Operationen (wie zum Beispiel normale Graphtraversierung) ohne Datenbankzugriff möglich sind (und damit auch die Existenz von AdjacenceArrayNode Instanzen entkoppelt von den Datenbankrepräsentanten sinnvoll), Instanzen von JPA Entity Bean Classes jedoch immer eine Zeile in der Tabelle darstellen.

Methoden

- *void setGraph(AdjacenceArrayGraph graph)* Definiert den zugehörigen Graphen. Dieser Setter muss aufgerufen werden, bevor *getNode()* funktionieren kann, da *getNode()* an *graph* deligiert.
- *Node getNode()* Gibt den zugehörigen Knoten zurück, indem an *graph.getNodeById()* deligiert wird.

2.4.1.4 Die Klasse CosimFeedback

Es hat sich gezeigt, dass eine Assoziation zwischen CosimFeedback und der abstrakten Edge-Klasse für die Berechnungen mancher Plugins sinnvoll ist, sodass nicht nur über Knoten- sondern auch über Kantenobjekte zwischen Datenbankklasse und internen Graphrepräsentation navigiert werden kann. Die Funktionsweise verläuft analog zu der in *CosimMember*.

Methoden

- *void setGraph(AdjacenceArrayGraph graph)* Definiert den zugehörigen Graphen. Dieser Setter muss aufgerufen werden, bevor *getNode()* funktionieren kann, da *getNode()* an *graph* deligiert.
- *Node getEdge()* Gibt ein Edge-Objekt mit den entsprechenden Endpunkten zurück, die mit *graph.getNodeById()* aus *peerId* und *serviceProviderId* ermittelt werden.

2.4.2 Änderungen im Package server.model.datatstructures

2.4.2.1 Die neue Klasse Admin

Die Entity-Bean Class Admin wurde nötig, damit mit JPA die eingerichteten Admins persistiert werden können. Instanzen dieser Klasse entsprechen also einzelnen Administratorenkonten, die aus Usernamen und Passwort bestehen. Die Klasse stellt entsprechende Getter- und Settermethoden bereit.

2.4.2.2 Die neue Klasse AppEntityManager

Diese Klasse stellt einen Singleton EntityManager für die applikationseigene Datenbank zur Verfügung. Die Notwendigkeit dafür konnte in der Entwurfsphase ohne Kentniss von der genauen Funktionsweise des Entity Managements in JPA nicht gesehen werden.

2.4.2.3 Die Klasse GraphModel

Das ermitteln der GraphModel-Instanz, die den Supergraphen enthält, wurde im Entwurf nicht vorgesehen.

Methoden

- *static GraphModel getSupergraphmodel()* Holt ein Model aus der Applikationseigenen Datenbank, das den Supergraphen enthält. Existiert solch ein Model nicht wird es erstellt und persistiert.

2.4.2.4 Die Klasse MeasureValueModel

Methoden

- *void normalize(double min, double max)* Diese Methode wurde im Zuge der Implementierung der Parameterangaben für Darstellungsarten von Zentralitätsmaße in GraphViz sinnvoll. Diese Darstellungsarten benötigen zu für die Darstellung eines konkreten Wertes eines Zentralitätsmaß einen konkreten Wert (zum Beispiel Rotanteil einer Farbe zwischen 0-255), deren Skalierungen aber im Allgemeinen unterschiedlich sind. Diese Funktion skaliert alle im Model befindlichen Werte auf einer Skala von *min* bis *max*. Die jewailigen Minimal- und Maximalwerte der MeasureStyles können über Gettermethoden in *client.datatstructures.MeasureStyle* ermittelt werden.

2.4.2.5 Das neue Interface GraphElement

Dieses Interface wird von Edge und Node implementiert und hatte technisch Gründe, die aus einigen generischen Klassen in server.controller.graphviz hervorgehen.

Methoden

- *String getName()* Name der Kante oder des Knotens, der dann eventuell als Beschriftung im Graphen wieder auftaucht.

2.4.2.6 Die Klasse Node

Das Auflösen der Vererbungsbeziehung zwischen Node und CosimMember (siehe 2.4.1.3) wurden folgende Methoden induziert, um die Navigation von Graphdatenstruktur zu Datenbankklassen weiterhin zu ermöglichen

Methoden

- *List<List<CosimFeedback>> getOutgoingCosimFeedbacks()* Gibt CosimFeedback-Instanzen für jede ausgehende Kante zurück.
- *CosimMember getCosimMember()* Gibt die dem Node entsprechende Datenbankklasseninstanz zurück.

2.4.2.7 Die neue generische Klasse Interval

Diese Klasse modelliert ein abgeschlossenes Interval, indem es die zwei Werte *min* und *max* speichert und entsprechende Getter- und Settermethoden bereitstellt. Semantische Überprüfung der gespeicherten Werte geschieht hierbei nicht.

2.4.2.8 Die neue generische Klasse IntervalSet

Diese Klasse ersetzt die im Entwurf verwendete Klasse *TimeConstraint* und veralgemeinert sie auf eine einfache Menge von Intervalen. Instanzen dieser Klasse werden wie im Entwurf zur Definition der zeitlichen Einschränkung eines Graphen verwendet, können jedoch zusätzlich zum Beispiel für Restriktionen auf Zentralitätsmaße wiederverwendet werden.

Methoden

- *boolean intersectionExists(..)* Überprüft, ob es eine Schnittmenge zwischen der Vereinigung aller gespeicherten Intervalle und der übergebenen Datenstruktur gibt. Diese Methode kann vor allem von Plugins benutzt werden, um zu ermitteln, ob der gerade berechnete Wert nicht den für sich gesetzten Restrictions wiederspricht.
- *static IntervalSet<Double> getByRestrictions(List<Restriction> restrictions, String name, Double min, Double max)* Erstellt eine IntervalSet-Instanz aus einer Liste von Restrictions, indem diese Liste nach Restrictions mit gegebenen Namen gefiltert wird und aus den verbliebenen Restriktionen so gut wie möglich Intervale gebaut werden. Die Angabe eines Minimal- und Maximalwertes ist hier erforderlich, um unvollständige Restriktionen (z.B. »<=30«) trotzdem sinnvoll interpretieren zu können.

2.4.2.9 Die Klasse AdjacenceArrayGraph

In dieser Klasse wurden Getter-Methoden für alle Attribute nötig, damit der Copy-Konstruktor implementiert werden konnte. Dies betrifft die Methoden *getNodesFrom()*, *getNodesTo()*, *getEdges()*, *getNodeCount()*, *getNodeIds()*, *getNodeIdRevLookup()* und *getEdgeRounds()*. Hier sei angemerkt, dass diese Methoden lediglich Kopien der Attribute zurückgeben und somit das Geheimnisprinzip bewahrt bleibt.

3 Implementierungsplan

2010-1

1 2 3 4 5 6

F S S M D M D F S S M D M D F S S M D M D F S S M D M D F S S M D M D F S S M D M D

Aufgabe	Status
~~Fehler #78~~: Graphrepräsentation	Erledigt 100%
~~Fehler #74~~: Implementierung von server.model	Erledigt 100%
~~Fehler #98~~: Stubs für Klassen mit externen	Erledigt 100%
~~Feature #80~~: Service Interfaces und Dummy	Erledigt 100%
~~Fehler #108~~: Notwendige Stubs in server.controller	Erledigt 100%
~~Feature #81~~: Stubs für Klassen mit externen	Erledigt 100%
~~Fehler #123~~: Einarbeitung in JPA	Erledigt 100%
~~Fehler #124~~: LoginPanel UI	Erledigt 100%
~~Fehler #125~~: GraphPanel UI	Erledigt 100%
~~Fehler #104~~: PluginManager	Erledigt 100%
~~Fehler #105~~: Plugin	Erledigt 100%
~~Fehler #126~~: GraphModel	Erledigt 100%
~~Fehler #113~~: CosimAccounting	Erledigt 100%
~~Fehler #115~~: CosimMember	Erledigt 100%
~~Fehler #117~~: CosimUsedPolicy	Erledigt 100%
~~Fehler #118~~: CosimFeedback	Erledigt 100%
~~Fehler #119~~: CosimSimulation	Erledigt 100%
~~Fehler #127~~: Applikationseigene JPA Schnittstelle	Erledigt 100%
~~Feature #82~~: minimales LoginPanel erstellen	Erledigt 100%
Fehler #106: SavedView	Zugewiesen 0%
~~Feature #85~~: NetworkPanel UI erstellen	Erledigt 100%
~~Fehler #122~~: InitPanel UI	Erledigt 100%
~~Fehler #79~~: Adjazenzarray Darstellung	Erledigt 100%
Implementierung - Bereitstellung der	Implementierung - Bereitstellung der Packageschnittstellen
~~Feature #86~~: MenuPanel UI erstellen	Erledigt 100%
~~Feature #88~~: GraphPanel erstellen	Erledigt 100%
~~Fehler #94~~: ServerRPC Logik für NetworkPanel schreiben	Erledigt 100%
~~Feature #90~~: implementierung von GraphInformation	Erledigt 100%
~~Feature #84~~: Logik für SocialViewEntryPoint schreiben	Erledigt 100%
~~Feature #89~~: GraphVizPanel implementieren	Erledigt 100%
~~Feature #87~~: MenuPanel Logik erstellen	Erledigt 100%
Implementierung - Fertigstellung der	Implementierung - Fertigstellung der Benutzeroberfläche
~~Feature #83~~: Logik für LoginPanel schreiben	Erledigt 100%
~~Fehler #102~~: Network	Erledigt 100%
Fehler #109: CachedGraphRequest	Zugewiesen 0%
~~Fehler #193~~: Integration von GraphViz	Erledigt 100%
Implementierung - Bereitstellung der	Implementierung - Bereitstellung der Pluginmanagementlogik
Fehler #107: CachedGraphManager	Zugewiesen 0%
~~Fehler #97~~: NetworkServiceImpl	Erledigt 100%
~~Fehler #95~~: AdminServiceImpl	Erledigt 100%
~~Feature #195~~: CosimUser	Erledigt 100%
~~Fehler #103~~: GraphCreator	Erledigt 100%
Fehler #96: SavedDisplayServiceImpl	Zugewiesen 0%
~~Fehler #91~~: Logik für Graph(Viz)Panel schreiben	Erledigt 100%
Implementierung - Das Package client.datastructures	Implementierung - Das Package client.datastructures
~~Fehler #92~~: InitPanel baisc einrichten	Erledigt 100%
~~Fehler #198~~: PluginManager.addPlugins wirft	Erledigt 100%
~~Fehler #129~~: Implementierung von	Erledigt 100%
~~Fehler #99~~: CredentialsVerifier	Erledigt 100%
~~Fehler #128~~: Implementierung von	Erledigt 100%
~~Fehler #100~~: AdminCredentialVerifier	Erledigt 100%
Implementierung - Integration von JPA	Implementierung - Integration von JPA
~~Fehler #101~~: GraphVizGraphDisplay	Erledigt 100%
Implementierung - Das Package server.controller	Implementierung - Das Package server.controller Phase 1
Implementierung - Das Package	Implementierung - Das Package server.model.datastructures
Implementierung - Das Package client.gui	Implementierung - Das Package client.gui
Implementierung - Das Package server.controller	Implementierung - Das Package server.controller Phase 2
Implementierung	Implementierung