HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

IBM

Framework für Web-basierte Darstellung von Time-Series Daten

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühjahrssemester 2014

Autoren:	Giuseppe Aquino, Simon Brouwer
Betreuer:	Prof. Dr. Markus Stolze
Projektpartner:	IBM Research - Zurich
Experte:	Remo Brunschwiler
Gegenleser:	Prof. Dr. Eduard Glatz

Technischer Bericht

Giuseppe Aquino & Simon Brouwer

1 Abstract

Aufgabenstellung Unter Time Series wird eine Folge von zeitabhängigen Messwerten oder Beobachtungen verstanden. Beispiele von Time Series sind die Entwicklung von Aktienkursen, Preisen oder Computer- bzw. Netzwerkperformanz. Time Series werden häufig als Linien-Diagramme visualisiert. Für die Darstellung von Time Series im Web existieren bereits eine Vielzahl von Bibliotheken welche es einfach machen Time Series Daten graphisch darzustellen. Der Industriepartner IBM Research - Zurich entwickelt verschiedene Systeme welche die Verarbeitung und Darstellung von Time Series voraussetzen. Für viele dieser Systeme werden unterschiedliche Libraries zur Darstellung (Plotten) der Time Series Daten verwendet. Das Ziel der vorliegenden Arbeit ist es ein Framework zu entwickeln welches das visualisieren der Time Series Daten vereinfacht und vereinheitlicht.

Ergebnisse Das Ergebnis der vorliegenden Arbeit stellt ein Client-, Server-basiertes Framework dar. Für die Visualisierung der Time Series Daten wurde auf Seite des Clients eine Plotting Library evaluiert auf welcher der Client basiert. Für die Entwicklung des Clients wurde JavaScript mit jQuery eingesetzt.
Die Server Seite dient als Proxy zwischen dem Client und dem Time Series Service. Für die Erweiterbarkeit des Proxys wurde das Spring Framework verwendet. Als Kommunikationsschnittstelle zwischen Client und Proxy dient eine REST-API, welche mit Jersey realisiert wurde.

Schlussfolgerung Die vorliegende Arbeit ermöglicht es dem Frontend Entwickler einer Time Series Applikation die Daten mit wenig Aufwand darzustellen. Gleichzeitig kann der Backend Entwickler seine Time Series Datenquelle für den Frontend Entwickler mit ein wenig Programmieraufwand zur Verfügung stellen.

2 Inhalt

3 Aufgabenstellung

Aufgabenstellung Bachelorarbeit Abteilung I, FS 2014
Giuseppe Aquino & Simon Brouwer

Reusable Library for Web-Based Visualization of Time Series Data

1 Auftraggeber und Betreuer

Praxispartner und Auftraggeber (nachfolgend IBM) diese Bachelorarbeit ist

IBM Research GmbH
IBM Research – Zurich
Säumerstrasse 4
CH-8803 Rüschlikon

Ansprechpartner Auftraggeber (formell):
Dr. Michael Osborne

Ansprechpartner Auftraggeber (informell):
Dr. Marc Stoecklin

Betreuer HSR:

Prof. Dr. Markus Stolze, Institut für Software
Weitere fachliche Unterstützung wird durch Silvan Gehrig (Assitent im IFS) geleistet

2 Ausgangslage

Time Series Daten sind eine Folge von zeitabhängigen Messwerten oder Beobachtungen. Diese Daten bilden die Grundlage für eine Vielzahl von Analysen und Anwendungen um die Vorgänge, Abhängigkeiten und auch Performanz eines Systems zu beobachen und zu verstehen. Als Beispiele für Anwendungsbereiche können Computer- oder Netzwerkperformanz (z.B. CPU Auslastung, Netzwerkflüsse), Aktienkurse, Preisentwicklung, Wetteranalyse (z.B. Temperaturwerte), oder demographische Statistiken (z.B. Bevölkerungszahl) angeführt werden. Bei einer Vielzahl von Anwendungen mit Time Series Daten ist deren Visualisierung und Analyse von zentralem Interesse.

3 Ziele der Bachelorarbeit

Im folgenden wird aus dem „Statement of Work" (engl) der IBM zitiert. Das vollständige „Statement of Work" ist im Anhang enthalten. Das Statement of Work ist bindend für diese Arbeit.

Es ist Aufgabe der Studenten mit dem Auftraggeber abzusprechen welche Dokumente (Delivearbles) auf Englisch erstellt werden müssen.

> 1 Scope of Work
>
> 1.1 The goal of this project is to develop a reusable library for web-based visualization of time series data that includes both client-side (integration and interaction) and server-side (data querying and interaction plugin API) components.
>
> 1.2 The project is composed of the following main activities:
>
> – Phase 1: Analyze requirements, design, and implement a web-based visualization library including flexible parametrization, generic transport format, various query models (e.g., depending on client-side interaction or server-side events)
> – Phase 2: Specification of a plugin API on client and server side that enable customized implementations of time series analytics and visualization.
> – Phase 3: Application of the library to two research projects including the assessment of the specific requirements (e.g., interaction, data flows, use cases) and a proof-of-concept implementation on top of real-world data.
>
> [...]
>
> 1.1 2.1.2 RAPPERSWIL's Responsibilities
>
> – RAPPERSWIL will provide a requirements analysis, code, and documentation;

4 Zusagen der IBM

- IBM will provide the students working on the project with background and specific goals of the project;
- IBM will designate the researcher(s) who will work with the students and provide guidance;
- IBM will deliver access to the research projects and their specific requirements.

5 Zur Durchführung

Die Arbeit startet am 17. Februar 2014 (KW 8). Abgabetermin des Berichtes an den HSR-Betreuer ist am 13. Juni 2014 um 12:00 Uhr (KW 24).

Mit dem HSR-Betreuer finden in der Regel wöchentliche Besprechungen statt. Zusätzliche Besprechungen sind nach Bedarf durch die Studierenden zu veranlassen.

Alle Besprechungen sind von den Studenten mit einer Traktandenliste vorzubereiten und die Ergebnisse in einem Protokoll zu dokumentieren, welches stets zugreifbar ist.

Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten. An Meilensteinen (gemäss Projektplan) sind einzelne Arbeitsresultate in vorläufigen Versionen abzugeben. Über die abgegebenen Arbeitsresultate erhalten die Studierenden ein Feedback. Eine definitive Beurteilung erfolgt aufgrund der am Abgabetermin abgelieferten Dokumentation. Die Evaluation erfolgt unter Nutzung des separat abgegebenen Kriterienkatalogs in Übereinstimmung mit den Kriterien zur BA Beurteilung.

6 Dokumentation

Über diese Arbeit ist eine Dokumentation gemäss den Richtlinien der Abteilung Informatik zu verfassen. Hierbei sind auch die Informationen im Dokument „Anleitung: Dokumentation Studien- und Bachelorarbeiten“ zu beachten, insbesondere hier der Anhang A: Minimalstandards für die Dokumentation von Studien- und Bachelorarbeiten mit Softwareentwicklung. Die zu erstellenden Dokumente sind im Projektplan festzuhalten. Alle Dokumente sind nachzuführen, d.h. sie sollen den Stand der Arbeit bei der Abgabe in konsistenter Form dokumentieren. Die Dokumentation ist vollständig auf CD/DVD in 2 Exemplaren abzugeben.

Titel der Arbeit, Namen der Studenten und Betreuer, Abstract der Arbeit und Auftraggeber werden von der HSR als Teil der Bachelorbroschüre und auf der Web-Site der HSR publiziert.

Neben der Bachelorarbeit ist im öffentlichen Wiki von Prof. Dr. M. Stolze eine Projektseite zu erstellen.

7 Datenschutz, Rechte

Es gelten die im Vertrag mit der IBM abgeschlossenen Regeln.

8 Weitere Regeln und Termine

Im Weiteren gelten die allgemeinen Regeln zu Bachelor und Studienarbeiten „Abläufe und Regelungen Studien- und Bachelorarbeiten im Studiengang Informatik“ (https://www.hsr.ch/Ablaeufe-und-Regelungen-Studie.7479.0.html)

Der Terminplan ist hier ersichtlich (HSR Intranet)
https://www.hsr.ch/Termine-Diplom-Bachelor-und.5142.0.html

9 Beurteilung

Eine erfolgreiche Bachelorarbeit zählt 12 ECTS-Punkte pro Studierenden. Für 1 ECTS Punkt ist eine Arbeitsleistung von ca. 25 bis 30 Stunden budgetiert. Entsprechend sollten ca. 350h Arbeit für die Bachelorarbeit aufgewendet werden. Dies entspricht ungefähr 25h pro Woche (auf 14 Wochen) und damit ca. 3 Tage Arbeit pro Woche pro Student.

Für die Beurteilung ist der HSR-Betreuer verantwortlich unter Einbezug des Beisitzers und allfälligem Feedbacks des Auftraggebers.

Die Bewertung der Arbeit erfolgt entsprechend der verteilten Kriterienliste.

Rapperswil, 11.4.2014

Prof. Dr. Markus Stolze
Institut für Software
Hochschule für Technik Rapperswi

4 Erklärung der eigenständigen Arbeit

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben.
- dass wir keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt haben.

Rapperswil, 13. Juni 2014

Giuseppe Aquino

Simon Brouwer

5 Management Summary

Ausgangslage

Unter Time Series wird eine Folge von zeitabhängigen Messwerten oder Beobachtungen verstanden. Beispiele von Time Series sind die Entwicklung von Aktienkursen, Preisen oder Computer- bzw. Netzwerkperformanz. Time Series werden häufig als Linien-Diagramme visualisiert. Für die Darstellung von Time Series im Web existieren bereits eine Vielzahl von Bibliotheken welche es einfach machen Time Series Daten graphisch darzustellen und Benutzern erlauben mittels Zooming und Panning die Daten genauer zu analysieren. Diese Bibliotheken übernehmen das Zeichnen und Skalieren von Zeitstrahlen sowie die Darstellung von einer oder mehrerer Time Series in einem interaktiven Diagramm. Im IBM Forschungslabor wurden bisher verschiedene Visualisierungs-Bibliotheken in unterschiedlichen Projekten eingesetzt. Dies macht die Wartung der Programme schwierig. Gleichzeitig zeigte sich, dass für die Anbindung der Visualisierungen an server-seitige Datenquellen stets ähnlicher Code entwickelt wurde. Mit diesem Projekt sollte eine Lösung erarbeitet werden welche eine standardisierte Visualisierungs-Bibliothek anbietet und es einfach macht diese mit server-seitigen Datenquellen zu verbinden.

Vorgehen

Nutzungsszenarien: Auf der Grundlage von zwei IBM-internen Projekten, welche Time Series Daten nutzen wurden Nutzungsszenarien ausgearbeitet. Die Nutzungsszenarien dienten als Grundlage für die Evaluation von Visualisierungs-Bibliotheken zur clientseitigen

Implementation des Frameworks.
Client Framework für die Darstellung der Daten: Der Client wurde basierend auf der ausgewählten Visualisierungs-Bibliothek entwickelt und bietet erweiterte Funktionalität zur Darstellung und Abfrage der Daten.

Servertechnologie: Weitere Time Series Systeme sollen möglichst einfach im Framework eingebunden werden können. Um dies zu ermöglichen, wurde ein Adapter-System, aufbauend auf Java und dem Spring Framework, entwickelt.

Ergebnisse

Die vorliegende Arbeit ermöglicht es dem Frontend Entwickler einer Time Series Applikation die Daten mit wenig Aufwand darzustellen. Gleichzeitig kann der Backend Entwickler seine Time Series Datenquelle für den Frontend Entwickler mit ein wenig Programmieraufwand zur Verfügung stellen. Zur Demonstration des Frameworks wurden zwei Adapter für die beiden IBM-internen Projekte geschrieben. Weitere Adapter können dynamisch in das Framework eingebracht werden.

6 Ausgangslage

Diese Arbeit beschäftigt sich mit der Visualisierung von Time Series Daten innerhalb des Webbrowser. Unter Time Series wird eine Folge von zeitabhängigen Messwerten oder Beobachtungen verstanden welche den Benutzer bei der Analyse eines Systems unterstützen. Typische Anwendungsbereiche für Time Series sind z.B. die Entwicklung von Aktienkursen, Preisen oder Computer- bzw. Netzwerkperformanz. Das Problem dass sich dabei ergibt ist die Visualisierung von Time Series im Browser. Bei der Vielzahl an Anwendungen, welche zur Auswertung Time Series einsetzen, ist es immer schwieriger eine einheitliche Implementierung anzubieten. Ziel der Bachelorarbeit ist es ein Framework zu kreieren, welches dem Entwickler eine API anbietet um die Implementierung mit möglichst wenigen Anpassungen zu vollenden.

6.1 Zielgruppe

Das Framework wird geschrieben um die Arbeit des Entwicklers bei der Visualisierung von Timeseries Daten zu erleichtern. Von der Optimierung profitieren in erster Linie die Entwickler welche sich mit der Programmierung von Web UIs befassen. In zweiter Linie bietet es dem Time Series Service Anbieter eine Möglichkeit seine Time Series Daten einfach zur Verfügung zu stellen.

6.1.1 Stakeholder Summary

Als Stakeholder wird der Entwickler gesehen, welcher die Aufgabe hat das Web UI für die Darstellung der Time Series zu programmieren. Diesem soll eine einfache und schnelle Lösung angeboten werden um die Time Series im Webinterface anzuzeigen.

Ausserdem soll der Time Series Service Anbieter seine Daten durch den Einsatz des Frameworks einfach zugreifbar machen können.

6.2 Heutige Praktiken und Lösungen

Aufgrund verschiedenartigen Time Series Daten, mit unterschiedlichen Datenformaten (JSON, XML etc.) ist die Darstellung oft statisch auf dem Client realisiert, weshalb die Time Series Daten müssen für den Graphen oft unterschiedlich aufbereitet werden um das Darstellen mit Hilfe der Plotting Library zu ermöglichen.

Wird das Datenformat der Datenquelle verändert, muss auch der Client auf die Änderung angepasst werden, indem die Aufbereitung der Daten angepasst wird.

Das Aufbereiten der Daten ins entsprechende Format fällt ausserdem oft zulasten des Clients welcher oft weniger Rechenleistung als der Server zur Verfügung hat.

Wie man in Abbildung 1 sehen kann werden die Daten vom Client abgerufen. Der Server retourniert die Daten. Sofern das Format der Server Antwort nicht Plotting Library konform ist, müssen die Daten für die Darstellung auf dem Client aufbereitet werden. Erst dann ist die Library in der Lage den Graphen darzustellen. Dieses Verfahren verschlechtert die Performanz auf dem Client und erhöht den Programmieraufwand des Web UI Entwickler, da dieser die Daten nach jeder Abfrage ins richtige Format bringen muss.

Abbildung 1: Aktuelle Kommunikation zwischen Client und Time Series Server

6.3 Motivation

Der Industriepartner IBM Research - Zurich entwickelt verschiedene Systeme welche die Verarbeitung und Darstellung von Time Series voraussetzen. Für viele dieser Systeme werden unterschiedliche Libraries zur Darstellung der Time Series Daten verwendet. Auch werden oft verschiedene Datenquellen für das Abrufen der Time Series Daten eingesetzt. Die Datenformate welche von den Time Series Services geliefert werden können variieren.

Während der Bachelorarbeit wird das zu produzierende Framework für zwei IBM interne Projekte analysiert. Dabei handelt es sich um ein Netzwerkanalyse Tool sowie ein Systemanalyse Tool. In der aktuellen Version der beiden Tools müssen die Time Series Daten immer explizit für die vom Tool verwendete Library angepasst werden damit diese dargestellt werden können.

Durch den Einsatz verschiedener Plotting Libraries und unterschiedlicher Datenformate ist es immer schwieriger eine einheitliche Implementation für die Visualisierung im Web UI anzubieten. Das Einarbeiten in ein neues System kann für den UI Entwickler somit zu einer schwerfälligen Arbeit werden, da dieser sich oft lange einlesen muss.

Durch das Framework soll das Aufbereiten der Daten für die Plotting Library vom Client entkoppelt werden, indem die Aufbereitung auf den Proxy verlegt wird.

Damit wird eine klare Separation of Concerns gewonnen durch die, der Client entlastet werden kann. Durch den Einsatz eines Proxys soll zudem möglich sein, ein einheitliches Datenformat für die Übertragung der Daten zu gewährleisten.

Für das Abfragen der Daten ist neu der Adapter zuständig, welcher die Kommunikation zwischen dem Time Series Service und dem Proxy umsetzt.

Der Proxy ermöglicht dem Entwickler mehrere Adapter hinzuzufügen um so verschiedene Time Series Services über den gleichen Proxy ansprechen zu können.

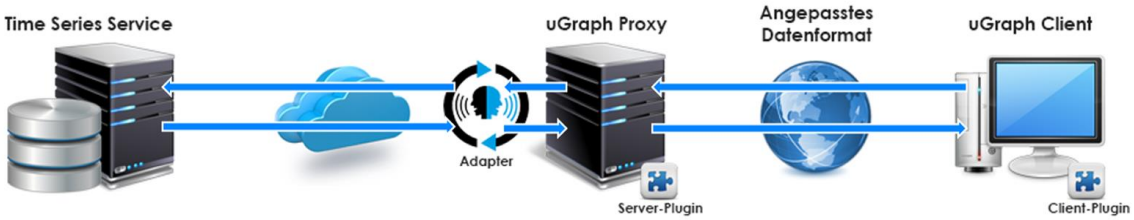


Abbildung 2 Kommunikation zwischen Client und Time Series Service mit Einsatz vom uGraph Framework

6.4 Einschränkungen

6.4.1 Lizenzierung der verwendeten Libraries

Externe Libraries müssen den Lizenzvorgaben des Auftraggebers IBM entsprechen. Der Einsatz einer Library muss immer zuerst mit dem Auftraggeber abgesprochen und genehmigt werden.

6.4.2 Client

Während der Bachelorarbeit soll eine Plotting Library evaluiert werden, welche als Grundlage für die clientseitige Implementation des Frameworks dient. Die Library welche dafür verwendet wird soll Open Source sein.

Für die Entwicklung des uGraph Client soll JavaScript eingesetzt werden.

6.4.3 Proxy

Der Proxy soll mit den eingesetzten Webcontainern der IBM kompatibel sein. Die beiden Tools, welche während der Bachelorarbeit analysiert werden setzen auf TomCat in der Version 7. Aus diesem Grund soll das Backend des Frameworks Java basierend sein.

7 Analyse

7.1 Domainanalyse

Aufgrund der von der IBM präsentierten Idee wurde eine Domänenanalyse erstellt (Siehe Abbildung 3). Die Domänenanalyse enthält neben den Client, den Server und der Time Series Service, weitere Komponenten welche mit dem Framework realisiert werden sollen.

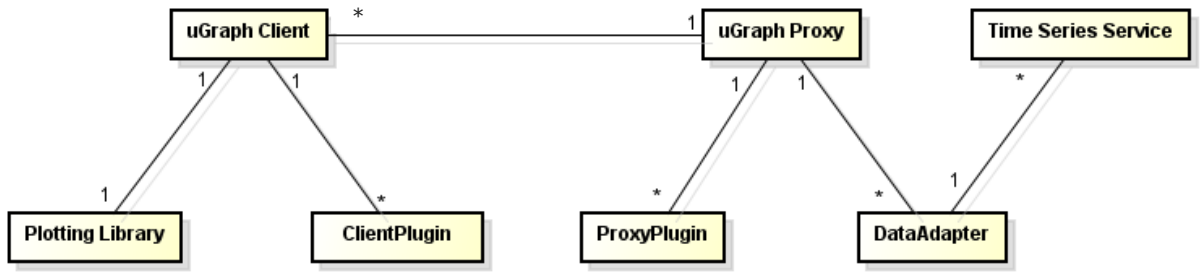


Abbildung 3 Analyse der Domäne

7.1.1 Komponenten

Folgend eine kurze Beschreibung der einzelnen Komponenten und deren Konzepte.

7.1.1.1 Plotting Library

Die Plotting Library ist eine JavaScript Library, welche es erlaubt Zeitreihendaten in einem Graphen darzustellen. Für diese Arbeit werden verschiedene bereits bestehende Library evaluiert. Die Funktionen der Plotting Library sollen nicht direkt benützt werden können, sondern über den uGraph Client aufgerufen werden. Somit soll eine einheitliche Implementation des Web UI ermöglicht werden, da für die Darstellung nur Funktionen des uGraph Clients nötig sind.

7.1.1.2 uGraph Client

Der uGraph Client ist ein Wrapper um die Plotting Library und erweitert diese um neue Funktionalität, welche die Zusammenarbeit mit dem Proxy ermöglichen und Features wie das dynamische Nachladen von Time Series Daten erlauben.

Der uGraph Client enthält die Kommunikationslogik welche das Abfragen der Time Series Daten ermöglicht. Dafür werden Ajax-Aufrufe eingesetzt.

7.1.1.3 ClientPlugin

ClientPlugins sind JavaScript Funktionen, welche beim uGraph Client registriert werden können. Die Plugins werden aufgerufen wenn der Client bereits Time Series Daten erhalten hat und können mit den abgefragten Daten arbeiten.

Ein ClientPlugin soll ermöglichen kleinere Berechnungen auf den abgefragten Time Series Daten noch vor der Darstellung durchzuführen, beispielsweise der Minima- oder Maxima-Wert eines Graphen zu berechnen.

7.1.1.4 Time Series Service

Der Time Series Service stellt eine Datenquelle dar, welche eine API für das Abfragen der Time Series Daten anbietet. Diese Datenquelle ist ein bereits existierendes System und somit nicht direkt Teil des uGraph Frameworks.

7.1.1.5 DataAdapter

Der DataAdapter ist die Komponente des Proxys, welche für die Kommunikation zwischen Time Series Service und Proxy zuständig ist. Erst durch den DataAdapter wird das Abfragen der Time Series Daten möglich. Damit eine Abfrage zustande kommen kann, muss der DataAdapter die API des Time Series Service benutzen.

Ein Proxy kann mehrere Implementationen eines DataAdapter besitzen. Dadurch können mehrere Time Series Service abgefragt werden. Weil verschiedene Time Series Services eingefügt werden können muss der DataAdapter identifiziert werden können. Dafür wird ein Identifier benötigt.

7.1.1.6 uGraph Proxy

Der uGraph Proxy bietet eine einfache REST API. Diese API wird für die Kommunikation zwischen Client und Proxy benötigt.

Für jede Anfrage des uGraph Cient ruft der uGraph Proxy den entsprechenden DataAdapter auf, welcher dann die Abfrage an den Time Series Service weiterleitet. Die Abgefragten Time Series Daten werden vom uGraph Proxy aufbereitet und an den uGraph Client zurückgesendet.

7.1.1.7 ProxyPlugin

Nachdem der uGraph Proxy die Daten einer Abfrage erhalten hat, können mit Hilfe der ProxyPlugins Berechnungen auf den abgefragten Daten ausgeführt werden. Beispielsweise könnte der Durchschnitt der Graphen berechnet werden.

ProxyPlugins sollen rechenintensive Berechnungen übernehmen um so den Client zu entlasten.

7.2 Funktionale Anforderungen

7.2.1 Personas

Die Personas stellen hypothetische Personen dar, welche später das System benutzen sollen. Anhand der Personas werden Nutzungsszenarien (Use Cases) geschrieben welche den Entwicklungsprozess unterstützen. Sie zeigen welche Nutzungsszenarien bzw. Features wirklich vom Benutzer benötigt werden oder gewünscht sein könnten. In diesem Abschnitt wird die Persona für dieses Projekt beschrieben.

Eugen Entwickler

Name	Eugen Entwickler
Alter	43
Funktion	Software Entwickler
Kenntnisse	Experte

Eugen Entwickler arbeitet seit einiger Zeit bei der IBM Research - Zurich in Rüschlikon. Er und sein Team entwickeln eine Web Applikation zur Überwachung von Netzwerkgeräten wie z.B. Switches und Routers.

Eugen arbeitet meist an seinem 15.5" Notebook auf welchem Windows 7 installiert ist. Als Browser ist die aktuellste Version des Firefox installiert welche er auch für das Testen der Web Applikationen benutzt.

Aufgrund seiner Arbeit als Software Entwickler ist er schon mit vielen Web-Interfaces in Berührung gekommen und hat auch schon selber solche entwickelt. Er ist auch vertraut mit dem Einsatz von Plotting-Libraries, würde jedoch wünschen sich nicht jedes Mal mit einer anderen Plotting Library befassen zu müssen, da dies immer wieder Einlese-Arbeit mit sich zieht.

7.2.2 Nutzungsszenarien

7.2.2.1 NS 01: Neue Datenquelle hinzufügen

Eugen hat gerade ein Projekt abgeschlossen und wurde bereits einem neuen Zugeteilt. Im neuen Projekt wird ebenfalls mit Time Series gearbeitet. Da Eugen im Frontend arbeitet ist seine Aufgabe eine Lösung für die Darstellung der Time Series Daten zu entwickeln. Eugen hat bereits im letzten Projekt mit uGraph gearbeitet. Durch den Einsatz des Frameworks blieb ihm viel Arbeit erspart. Das neue Projekt will er daher wieder mithilfe von uGraph entwickeln. Als er versucht ein Plot im UI zu machen, bemerkt er jedoch dass der uGraph Proxy keine Daten zurückgeliefert hat und deshalb die Darstellung der Time Series Daten ausfällt. Ihm fällt ein, dass es dafür noch ein DataAdapter für die neue Datenquelle auf den uGraph Proxy braucht. Eugen schreibt den neuen DataAdapter und registriert diesen auf dem Proxy. Die

neue Datenquelle ist somit eingebunden und kann nun vom uGraph Proxy angesprochen werden. Die Abfragen werden nun richtig ausgeführt.

7.2.2.2 NS 02: Neues Server-Plugin hinzufügen

Eugen ist gerade bei der Entwicklung eines Graphen. Gerne würde er dem Benutzer der Webapplikation die Möglichkeit bieten den Mittelwert eines Graphen darzustellen. Da die Time Series Daten viele Messpunkte enthalten, wäre das berechnen des Mittelwertes auf dem uGraph Client zu rechenintensiv. Er beschliesst deshalb diese Berechnung auf dem uGraph Proxy auszulagern. Dafür schreibt er ein ProxyPlugin welches erlaubt direkt den Mittelwert einer bestimmten Time Series zu berechnen. Das ProxyPlugin registriert er anschliessend auf dem uGraph Proxy. Nun kann vom uGraph Client aus, direkt der Mittelwert einer bestimmten Time Series abgefragt werden.

7.2.2.3 NS 03: Graphen an Bedürfnisse der View anpassen

Bei der Entwicklung eines Graphen möchte Eugen einen bestimmten Time Series Service abfragen. Dieser Service liefert nur Daten in sehr grossen Abständen, weshalb er die Zeitformatierung auf der X-Achse anpassen möchte. Eugen passt dafür die Darstellungsoptionen an, so dass nicht mehr Stunden sondern Tage angezeigt werden. Nun werden auf der X-Achse Tage statt Stunden angezeigt nun wirkt der Graph übersichtlicher.

7.2.3 Use Cases

7.2.3.1 Use Case Diagramm

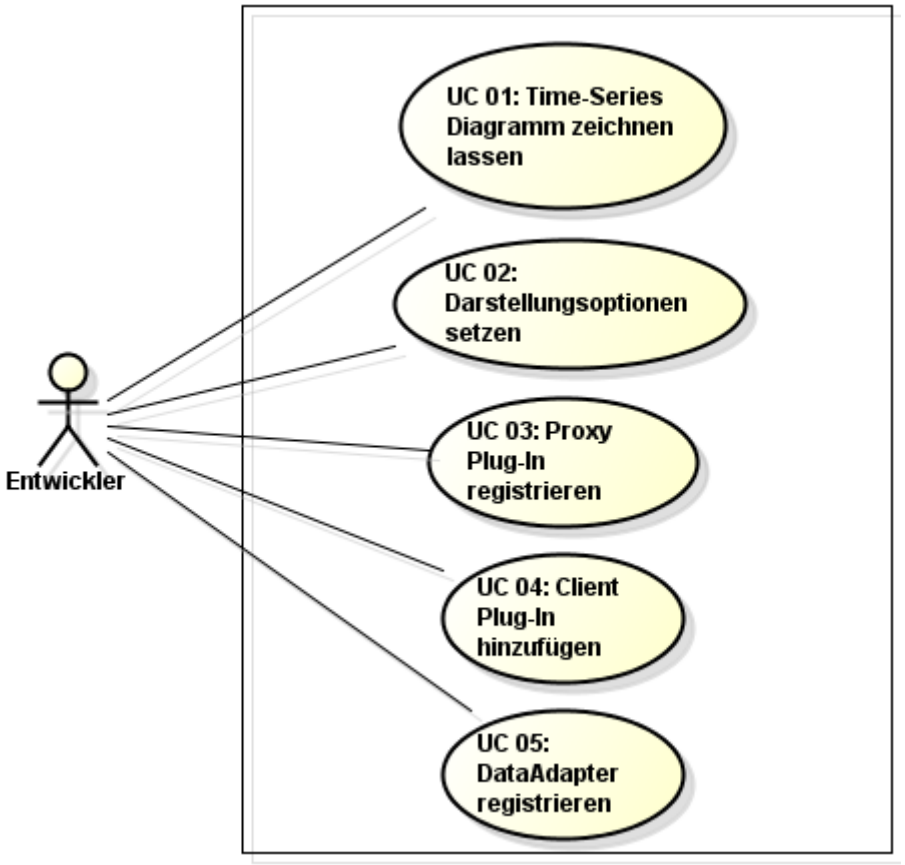


Abbildung 4 Use Case Diagramm

7.2.3.2 Use Cases (Brief)

7.2.3.2.1 UC 01: Time Series Diagramm zeichnen lassen

Der Entwickler ruft die Funktion des uGraph Frameworks auf um einen Graphen zu zeichnen. Dabei übergibt er die benötigten Parameter um den gewünschten Graph zu zeichnen.

7.2.3.2.2 UC 02: Darstellungsoptionen setzen

Der Entwickler will das bestehende Diagramm ändern. Dafür verändert er die Darstellungsoptionen. Daraufhin wird das Diagramm vom uGraph Framework entsprechend angepasst.

7.2.3.2.3 UC 03: ClientPlugin registrieren

Der Entwickler des Web UI möchte nach dem Abrufen der Time Series Daten weitere, kleinere Berechnungen auf den Daten durchführen. Dafür schreibt er ein ClientPlugin und registriert diesen auf dem uGraph Client

7.2.3.2.4 UC 04: ProxyPlugin registrieren

Der Entwickler will die abgefragten Time Series Daten weiter für die Darstellung aufbereiten und registriert auf dem uGraph Proxy ein Plugin welches die gewünschten Berechnungen durchführt.

7.2.3.2.5 UC 05: DataAdapter registrieren

Ein Entwickler will mit Hilfe des uGraph Frameworks ein neuer Time Series Service ansprechen, dafür registriert er ein DataAdapter auf dem uGraph Proxy.

7.3 Nicht Funktionale Anforderungen

7.3.1 Funktionalität

7.3.1.1 Richtigkeit

Bei Abfragen der Time Series Daten von einem Time Series Service soll die Anfrage vom Proxy richtig weitergeiltet werden, so dass die abgefragten Time Series Daten stets der Anfrage vom uGraph Client entsprechen.

7.3.1.2 Interoperabilität

Die auf dem uGraph Proxy laufende REST API muss auf einem TomCat 7 oder neuer lauffähig sein.

Der uGraph Client soll mit dem Firefox Browser mit Version ab 24.3.0 kompatibel sein.

Mit Hilfe eines DataAdapter soll es möglich sein beliebige Time Series Services mit dem entwickelten Framwork zu verwenden.

7.3.2 Zuverlässigkeit

7.3.2.1 Fehlertoleranz

Bei einer fehlerhaften Abfrage im uGraph Framework, soll der uGraph Client vom uGraph Proxy mit der entsprechenden Fehlermeldung informiert werden. Das Framework soll weiterhin in der Lage sein weitere Anfragen zu stellen.

7.3.3 Benutzbarkeit

7.3.3.1 Erlernbarkeit

Das System wird von Personen mit guten technischen Kenntnissen verwendet. Diese sollen mit Hilfe der API-Dokumentation in der Lage sein das System innerhalb einer Woche verwenden zu können.

7.3.4 Effizienz

7.3.4.1 Zeitverhalten

Ein Graph soll, bei Verwendung des Frameworks innerhalb eines lokalen Netzes, in weniger als 3 Sekunden gerendert werden können.

7.3.5 Wartbarkeit

7.3.5.1 Modifizierbarkeit

Das uGraph Framework soll erweiterbar gestaltet sein. Die Einbindung solcher Erweiterungen, in Form von DataAdapter, ProxyPlugin und ClientPlugin soll innerhalb von 15 Minuten möglich sein.

Das uGraph Framework soll einfach zu migrieren sein. Durch einfaches kopieren soll der Server sowie die Client innerhalb 30 Minuten in eine neue TomCat Umgebung migriert werden können.

7.3.5.2 Testbarkeit

Für das Testen des uGraph Frameworks soll keine Internetverbindung bzw. Netzwerkverbindung nötig sein. Benötigte Testdaten sollen mithilfe von Mocking Objekte simuliert werden können. Die Implementierung einer Mock-Klasse soll durch ein Interface erleichtert werden und nicht länger als eine halbe Stunde benötigen.

Um das System einfach und schnell testen zu können werden für sämtliche wichtigen Funktionen Unit-Tests geschrieben. Damit soll nach einer Änderung im Code, mit minimalem Aufwand erkannt werden ob das System noch lauffähig ist oder nicht. Die Durchführung der Tests soll nicht länger als 5 Minuten in Anspruch nehmen.

7.3.6 Übertragbarkeit

7.3.6.1 Installierbarkeit

Um das System in einer bestehenden Umgebung zu installieren und zu verwenden, soll nicht mehr als 30 Minuten benötigt werden. Dies gilt nur für die Installation und beinhaltet nicht die Implementierung eines DataAdapters oder Proxy- bzw. ClientPlugin.

7.4 Feldforschung

Vor der Evaluation der Plotting Library wurden zwei Sitzungen mit den Teammitgliedern der zu analysierenden IBM Projekte durchgeführt. Dabei lag der Fokus auf den Anforderungen welche die Plotting Library abdecken soll.

Für die Tools wurden folgende Plotting Libraries verwendet:

- Flot
- jQPlot

Bei beiden Plotting Libraries fehlten den Teams gewisse Funktionen, welche nun im neuen Framework unterstützt werden sollten. Diese fehlenden Funktionen wurden in den Beiden Sitzungen besprochen und die passenden Anforderungen daraus aufgestellt.

7.4.1 Anforderungen an Plotting Library (Systemanalyse Tool)

Für das Systemanalyse Tool ergaben sich nach einem Brainstorming folgende Anforderungen:

- Verschiedene Einheiten in einem Diagramm anzeigen
- Plugin Unterstützung
- Graphischen Zoomen und Schwenken
- Preloading der Time Series Daten
- Highlighting eines Graphen bei Anzeige mehrerer Graphen in einem Diagramm
- Konfigurierbare Legende
- Achsenwerte sollen formatierbar sein
- Events im Graphen vermerken (Annotation um einen bestimmten Zeitpunkt zu markieren)
- Min- und Maxgraphen werden angezeigt und der Bereich zwischen den Graphen wird schraffiert
- Wenn mit der Maus über ein Datenpunkt im Diagramm gefahren wird, wird der Wert des Punktes angezeigt (Detail on Demand)

7.4.2 Anforderungen an Plotting Library (Energieverwaltungs Tool)

Anfänglich war geplant das Energieverwaltungs Tool, welches bereits aus der Studienarbeit bekannt war für uGraph zu Analysieren und für die Beispiel-Implementation zu verwenden. Leider war die Implementation nicht mehr möglich, da das Tools nicht mehr in Betrieb war. Die erarbeiteten Anforderungen des Energieverwaltungs Tools wurden trotzdem für die Evaluation der Plotting Library verwendet.

Folgende Anforderungen wurden erarbeitet:

- Vorhersagen von Time Series Daten im Graph darstellen
- Graphischen Zoomen und Schwenken
- Events im Graphen vermerken (Annotation um einen bestimmten Zeitpunkt zu markieren)
- Min- und Maxgraphen werden angezeigt und der Bereich zwischen den Graphen wird schraffiert
- Verteilfunktion über die Werte. Kleiner Graph (um 90 Grad gedreht) auf der Seite des eigentlichen Diagramms, welcher die Verteilfunktion anzeigt
- Verteilfunktion über die Zeit. Kleiner Graph oberhalb des eigentlichen Diagramms, welcher zeigt wann wie viele Werte erfasst wurden

7.4.3 Schlussfolgerung

In einer abschliessenden Sitzung mit dem Industriepartner wurden die Anforderungen nochmals überprüft. Es wurde festgelegt, dass die Erkenntnisse aus den beiden Sitzungen mit den Referenzprojekten in die Evaluation einbezogen werden. Dabei sind nur jene Anforderungen gemeint, welche zu einer Verbesserung der aktuellen Implementierung führen und im Zeitrahmen der Bachelorarbeit umgesetzt werden können.

8 Umsetzung

8.1 Evaluation der Plotting Library

Die Evaluation der Plotting Library wird in zwei Iterationen aufgeteilt. Dabei besteht immer die gleiche Gewichtung der einzelnen Kriterien:

- 1: unwichtig
- 2: wichtig
- 3: sehr wichtig

Die Kriterien sind zusammengesetzt aus gewünschten Funktionen der beiden Referenzprojekte (das Systemanalyse Tool und das Energieanalyse Tool), den allgemeinen Einschränkungen (6.4 Einschränkungen, Seite 12) und für die Architektur des Frameworks relevante Eigenschaften, wie z.B. das unterstützte Datenformat der Plotting Library.

8.1.1 Iteration 1

Für die Iteration 1 wurden die Kriterien der Tabelle 1 gewählt. Durch diese erste grobe Eingrenzung sollten möglichst viele Libraries eliminiert werden um so, in der zweiten Iteration, die übrigen Libraries genauer analysieren zu können. Speziell in dieser Iteration ist der Umgang mit unerfüllten Kriterien. Da es sich in dieser Phase nur um Ausschlusskriterien handelt ist die Gewichtung überall 3. Ein nichterfülltes Kriterium hat zur Folge, dass die Library nicht weiter untersucht wird.

Nr.	Titel	Beschreibung	Gewichtung
K1.1	Achsenformatierung	Achsenwerte und Formatierung sollen konfigurierbar sein. Z.B. 1.0 statt 1000’000’0000 um die View einheitlich zu gestalten. Dieses Kriterium wurde aufgenommen weil der Kunde bereits Probleme in der Darstellung vom jetzigen System hatte.	3
K1.2	Open Source	Die Plotting Library muss Open Source. Dieses Kriterium ist eines der Einschränkungen für das uGraph Framework.	3
K1.3	Nicht Viral	Die Lizenz der Plotting Library muss so ausgelegt sein, dass der restliche Code nicht auch Open Source sein muss (nicht Viral). Dies gilt um die internen Richtlinien der IBM zu erfüllen.	3
K1.4	Kompatibilität	Die Plotting Library darf nicht mit jQuery oder anderen vom uGraph Framework verwendeten Libraries und Frameworks im Konflikt stehen.	3
K1.5	Time Sries	Die Plotting Library muss Time Series Diagramme unterstützen. Andere	3

		Typen von Diagrammen sind erwünscht jedoch für das Projekt nicht relevant.	
K1.6	JavaScript	Die Plotting Library muss mit JavaScript kompatibel sein. Dies soll die Implementierung vom uGraph Client erleichtern, welcher in JavaScript geschrieben werden soll.	3
K1.7	Zoomen und Schwenken	Graphisches Zoomen und Schwenken: Auf dieses Feature wurde von beiden Referenzprojektteams sehr viel Wert gelegt.	3

Tabelle 1 Evaluationskriterien Iteration 1

8.1.2 Iteration 2

Diese Anforderungen dienen dazu aus der näheren Auswahl der vorherigen Iteration eine Library auszuwählen, welche dann für das Projekt verwendet wird.

In dieser Iteration müssen nicht sämtliche Kriterien erfüllt werden. Stattdessen werden die erfüllten Anforderungen gezählt und mit der jeweiligen Gewichtung multipliziert, diejenige Plotting Library welche am meisten Punkte erhält wird nach Rücksprache mit dem Partner IBM für das uGraph Framework verwendet.

Nr.	Titel	Beschreibung	Gewichtung
K2.1	Performanz	Viele Graphen in einem Diagramm anzeigen, ohne Performanz Einbussen.	3
K2.2	Hover	Wenn mit der Maus auf einen Datenpunkt gezeigt wird dessen Wert angezeigt.	2
K2.3	Aktualisierung	Dynamische Aktualisierung des Graphen.	3
K2.4	Verschiedene Einheiten	Graphen mit unterschiedlichen Einheiten im gleichen Diagramm anzeigen.	2
K2.5	"Band"-Darstellung	Min- und Max-Mittelwertgraphen werden angezeigt und der Bereich zwischen Min und Max wird schraffiert	1
K2.6	JSON Unterstützung	Die Library soll das JSON Datenformat unterstützen.	1
K2.7	Plugin Unterstützung	Erweiterungen der Plotting Library sind bereits vorhanden oder können zu einem späteren Zeitpunkt in Form von Plugin hinzugefügt werden.	1
K2.8	Graph Highlighting	Wenn 2 oder mehrere Graphen sich überlappen sollte man einen Graphen hervorheben können.	2
K2.9	Events	Events im Graph vermerken, damit Anomalien und Korrelationen schneller visuell festgestellt werden können.	2

K2.10	Gesamtes Diagramm drehen	Das gesamte Diagramm um 90 Grad drehen. Dies wurde gewünscht um Beispielsweise eine Visualisierung der Verteilfunktion der Time Series Daten neben dem eigentlichen Diagramm anzuzeigen.	1
K2.11	API	Die Library soll eine gute API Dokumentation bereitstellen. Dieses Kriterium wird als wichtig erachtet weil die weitere Verwendung der Library für den Entwickler möglichst einfach sein soll.	3

Tabelle 2 Evaluationskriterien Iteration 2

8.1.3 Untersuchte Libraries

Die Liste der Plotting Libraries, welche für die Evaluation gewählt wurden, stammt aus zwei verschiedenen Internet-Quellen[12].

Zusätzlich wurden auch die Libraries der Referenzprojekte mit in den Evaluationsprozess aufgenommen.

- jqPlot
- Flot

Es wurden nur jene Libraries in den Evaluationsprozess aufgenommen, welche eine Open Source Lizenz haben.

Name	Link	Lizenz
Google Charts	https://developers.google.com/chart/	Creative Commons Attribution 3.0 License
gRaphaël	http://g.raphaeljs.com/	MIT Lizenz
RGraph	http://www.rgraph.net/	MIT Lizenz
jQuery Sparklines	http://omnipotent.net/jquery.sparkline/#s-about	BSD 3 Lizenz
Chart.js	http://www.chartjs.org/docs/	MIT Lizenz
Dygraphs	http://dygraphs.com/	MIT Lizenz
flotr2	http://humblesoftware.com/flotr2/	MIT Lizenz
PlotKit	http://www.liquidx.net/plotkit/	BSD Lizenz / Apache Lizenz
Rickshaw	http://code.shutterstock.com/rickshaw/	MIT Lizenz
NVD3	http://nvd3.org/index.html	Apache Lizenz, Version 2.0
dc.js	http://nickqizhu.github.io/dc.js/	Apache Lizenz, Version 2.0
xCharts	http://tenxer.github.io/xcharts/	https://github.com/tenXer/xcharts/blob/master/LICENSE

[1] [WikJSPL]
[2] [DivJSPL]

YUI Charts	http://yuilibrary.com/yui/docs/charts/	BSD Lizenz
jqPlot	http://www.jqplot.com/	GPL und MIT Lizenz
Flot	http://www.flotcharts.org/	MIT Lizenz

Tabelle 3 Liste der evaluierten Plotting Libraries

8.1.4 Entscheidungsmatrix Iteration 1

Folgend die Entscheidungsmatrix für die erste Iteration.

Name	K1.1 * 3	K1.2 * 3	K1.3 * 3	K1.4 * 3	K1.5 * 3	K1.6 * 3	K1.7 * 3	Total
Google Charts	1	0	1	1	1	1	1	18
gRaphaël	0	1	1	1	1	1	0	15
RGraph	1	1	1	1	1	1	0	18
jQuery Sparklines	0	1	1	1	1	1	0	15
Chart.js	1	1	1	1	1	1	0	18
Dygraphs	1	1	1	1	1	1	1	21
flotr2	1	1	1	1	1	1	1	21
PlotKit	0	1	1	1	0	1	0	12
Rickshaw	1	1	1	1	1	1	1	21
NVD3	1	1	1	1	1	1	1	21
dc.js	1	1	1	1	1	1	1	21
xCharts	0	1	1	1	1	1	0	15
YUI Charts	1	1	1	1	1	1	0	18
jqPlot	0	1	1	1	1	1	1	18
Flot	0	1	1	1	1	1	1	1

Tabelle 4 Entscheidungsmatrix Iteration 1

Legende

- Wird weiter analysiert
- Wird nicht weiter analysiert

8.1.5 Iteration 2

Die Auswertung der Evaluation für die Iteration 2 ist in der Tabelle 5 veranschaulicht. Folgend eine Beschreibung der einzelnen Plotting Libraries und die Begründung für die Ausscheidung.

8.1.5.1 Dygraphs

Dygraphs war von Anfang an bereits eines der favorisierten Plotting Libraries. Optisch gesehen ist die Library sehr wandlungsfähig und bietet zudem sehr viel Funktionalität an. Ein Manko in der Library war die Unterstützung von JSON als Dateiformat, stattdessen wird in Dygraph mit CSV gearbeitet oder wie in den anderen Libraries mit einem normalen JavaScript Array, dieses Kriterium war jedoch nicht als wichtig angesehen worden da die Daten später vom uGraph Proxy in das richtige Format gebracht werden können. Das drehen des Diagramms um 90 Grad wurde von keiner der Libraries unterstütz, weshalb dieses Kriterium die Wertung nicht beeinflusste.

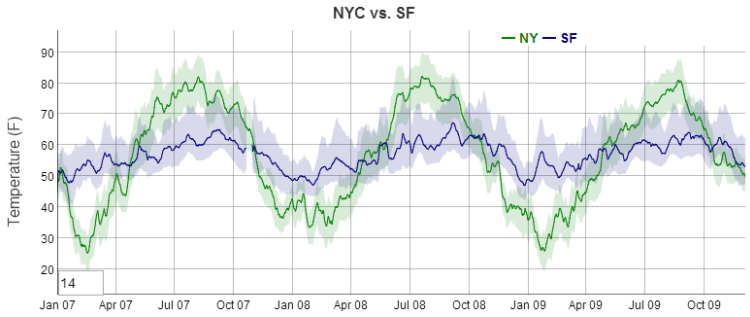


Abbildung 5 Beispiel Dygraphs

8.1.5.2 flotr2

flotr2 ist ein Derivat der Prototype basierten Plotting Library flotr[3]. Dieses Derivat basiert jedoch nicht auf Prototype sondern auf jQuery. Leider konnte diese Plotting Library, wie in Tabelle 5 zu sehen ist, viele der gewünschten Punkte nicht abdecken. So war z.B. die gewünschte "Band"-Darstellung gar nicht möglich. Eines der wichtigen Kriterien, welches hauptsächlich das Systemanalyse Tool betrifft, war das Highlighting des Graphen. Im Systemanalyse Tool werden bis zu 360 Graphen auf einem Diagramm angezeigt, was schnell zu einer unübersichtlichen Darstellung führen kann. Aus diesem Grund wurde dieses Kriterium auch etwas höher gewertet. flotr2 ist im gesamtpacket keine schlechte Library, sie ist relativ modern aufgebaut und bietet ebenso auch JSON Unterstützung. Für die Anforderungen dieses Projektes wurde sie dennoch nicht als geeignet gewertet.

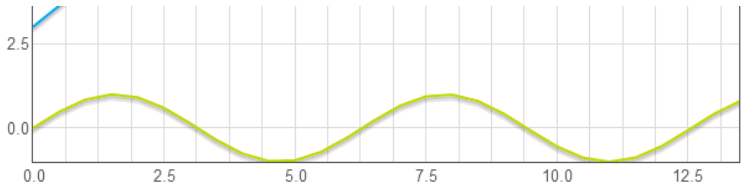


Abbildung 6 Beispiel flotr2

[3] [FloLIB]

8.1.5.3 Rickshaw

Die auf d3.js basierende Rishshaw Library schloss in der Evaluation punktemässig gleich gut wie Dygraphs ab. Sie hat in der Evaluation die meisten Kriterien erfüllen können und war mit Dygraph einer der beiden finalen Kandidaten. Die Library bietet eine gute Dokumentation der API. Viele der Funktionen sind anschaulich präsentiert und einfach nachvollziehbar durch Beispiele. Trotz der guten Resultate im Test musste gegen Rickshaw entschieden werden, da für das Zooming und Panning ein zusätzlicher Selektor-Diagramm nötig ist, in welchem dann der darzustellende Bereich ausgewählt und verschoben werden kann. Dies ist umständlicher als direkt im Diagramm den Bereich zu selektieren und nimmt mehr Platz in Anspruch. Aus diesen Gründen musste vom Gebrauch von Rickshaw abgesehen werden.

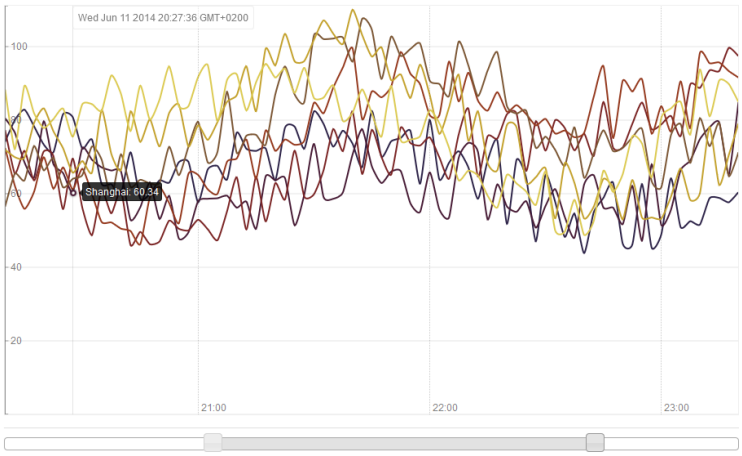


Abbildung 7 Beispiel Rickshaw

8.1.5.4 NVD3

NVD3, welches ebenfalls auf der d3.js Library aufbaut, hat im Test am schlechtesten abgeschlossen. Ausschlaggebend war hier vor allem die nicht vorhandene API-Referenz sowie auch sonst ungenügende Dokumentation. Ausserdem wurden wichtige Anforderungen, wie das Hervorheben eines Graphs oder das vermerken von Events im Diagramm, nicht unterstützt.

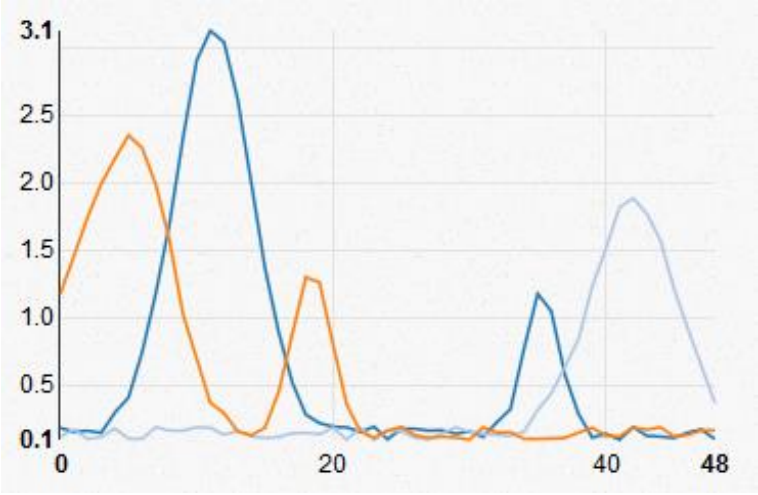


Abbildung 8 Beispiel NVD3

8.1.5.5 dc.js

Die dritte Bibliothek welche auf d3.js basiert war dc.js. Diese Plotting Library überzeugt mit einer übersichtlichen API-Referenz, hinkt jedoch bei anderen Kriterien hinterher; so ist es nicht möglich mehrere Y-Achsen mit unterschiedlichen Einheiten darzustellen. Neben d3.js hängt dc.js auch von der crossfilter[4] Library ab, diese wird dazu benötigt die darzustellenden Daten für die Darstellung im Diagramm aufzubereiten, was die Verwendung der Library unnötig verkompliziert.

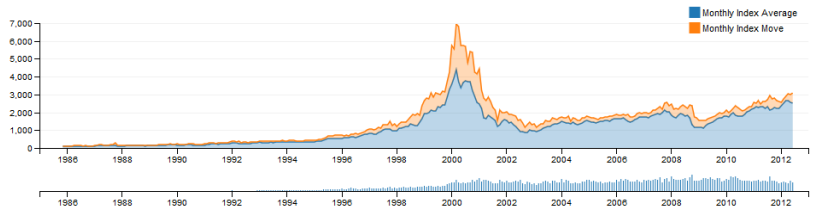


Abbildung 9 Beispiel dc.js

[4] [SquLIB]

8.1.6 Entscheidungsmatrix Iteration 2

Folgend die Entscheidungsmatrix für die erste Iteration.

Name	K2.1 * 3	K2.2 *2	K2.3 * 3	K2.4 * 2	K2.5 * 1	K2.6 * 1	K2.7 * 1	K2.8 * 2	K2.9 * 2	K2.10*1	K2.11*2	Total
Dygraphs	1	1	1	1	1	0	1	1	1	0	1	19
flotr2	1	1	1	1	0	1	1	0	0	0	1	15
Rickshaw	1	1	1	1	1	1	0	1	1	0	1	19
NVD3	1	1	1	1	1	1	0	0	0	0	0	12
dc.js	1	1	1	0	1	1	0	1	0	0	1	15

Tabelle 5 Entscheidungsmatrix Iteration 2

Legende

- Wird weiter analysiert
- Wird nicht weiter analysiert

8.1.7 Beschluss

Nach der Evaluation wurden die zwei Libraries mit der höchsten Wertung der IBM Präsentiert - Dygraphs und Rickshaw. Die gestellten Kriterien wurden nochmals gemeinsam mit dem Industriepartner IBM überprüft und sich für Dygraphs entschieden. Die Unterstützung von JSON wurde wie bereits erwähnt nicht hoch gewertet und da Dygraphs ohnehin Plugins unterstützt, wäre eine nachträgliche Erweiterung durchaus möglich. Ausserdem wurde es als ungünstig empfunden, dass bei Rickshaw für das Zooming und Pannign ein weiteres Selektor-Diagramm nötig ist. Aus diesem Grund hat sich der Partner IBM für Dygraphs entschlossen.

8.2 Prototyp & Proof of Concept

Das von der IBM vorgestellte Framework enthielt Konzepte mit welchen das Team während dem Studium noch keine Erfahrungen gemacht hatte. Aus diesem Grund wurden diese Konzepte mit der Entwicklung eines Prototyps überprüft.

8.2.1 Konzept DataAdapter

Der uGraph Proxy soll erweiterbar gestaltet sein und Time Series Services sollen einfach und ohne Änderung des Proxys hinzugefügt werden können. Um das zu realisieren, kann ein DataAdapter in einem externen JAR implementiert werden und mit Hilfe eines Interfaces auf dem Proxy eingebunden werden.

8.2.2 Laden der externen DataAdapter JARs zur Laufzeit

Für das Laden der Adapter wurde JNDI eingesetzt. Mit Hilfe von JNDI wurde der Zugriff auf das Dateisystem realisiert. Dadurch konnte auf die externen JARs zugegriffen werden. Durch den gewonnenen Zugriff auf das Dateisystem konnte mit Hilfe eines URLClassloaders das entsprechende JAR geladen werden.

Durch den Einsatz von JNDI ist es nötig den genauen Pfad zum JAR zu kennen, welches die Implementierung des DataAdapters enthält.

8.2.3 Konzept uGraph Client

Auch für den uGraph Client wurden Proof of Concepts gemacht, hauptsächlich um die Zusammenarbeit mit der Plotting Library zu überprüfen und die notwendigen Parameter für einen uGraph Client Aufruf zu bestimmen. Um die Kommunikation zwischen uGraph Client und uGraph Proxy einheitlich zu halten wurden verschiedene Datenformate evaluiert. Das Finden des finalen Datenformats war ein langer Prozess und wird aus diesem Grund im Punkt 8.3,

Evaluation der Kommunikation zwischen Client und Proxy, auf Seite 34 beschrieben.

8.2.4 Prototyp

Für den Prototyp wurde ein möglicher Ablauf bestimmt und mit einem Sequenzdiagramm abgebildet. Wie man in Abbildung 10 sehen kann wird der DataAdapter aus dem JAR erst bei einer Abfrage auf dem uGraph Proxy instanziiert, der Vorteil dabei ist, dass ein DataAdapter nur dann instanziiert wird, wenn er benötigt wird. Der DataAdapter fragt dann die Time Series Daten von der Datenquelle ab.

Für den Prototyp wurde als Datenquelle eine Sample-Abfrage aus dem Systemanalyse Tool der IBM, in JSON Format abgefragt. Diese Datei war auf einem Apache Webserver abgelegt und diente als Time Series Service für den Prototyp.

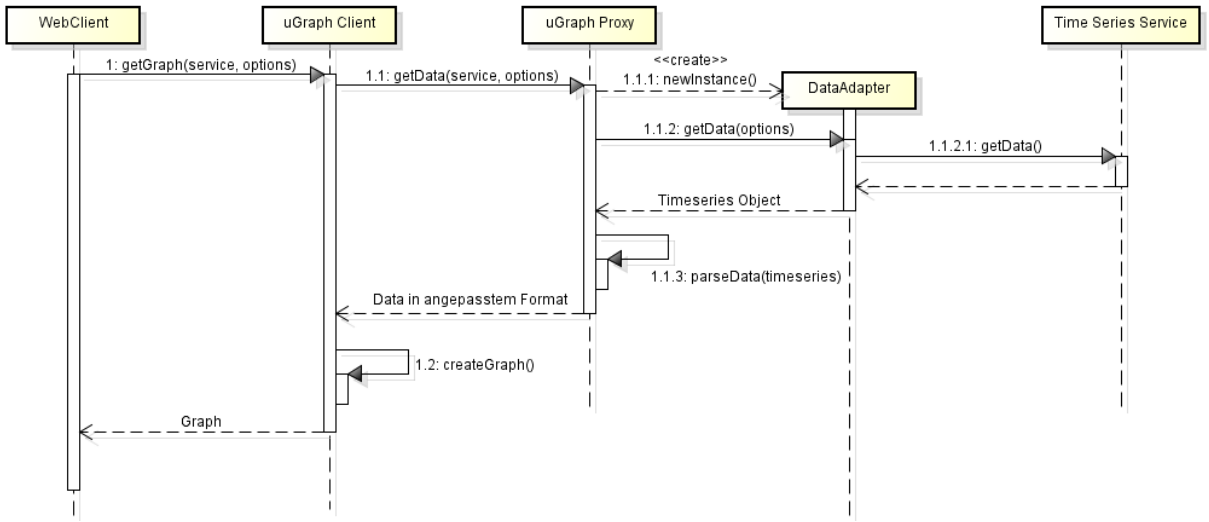


Abbildung 10 Sequenzdiagramm Request Prototyp

8.3 Evaluation der Kommunikation zwischen Client und Proxy

Das Übertragungsformat für die Abfrage von Time Series Daten wurde über die gesamte Entwicklung des Frameworks immer wieder überarbeitet. Die Parameter für die Abfrage von Time Series Daten konnten erst im Verlauf der Arbeit konkretisiert werden, weshalb diese von Version zu Version variieren.

8.3.1 Parametrisierung der Zeitspannenangabe für Time Series Abfragen

Eine Time Series ist immer über eine Zeitspanne abfragbar, das heisst es besitzt stets ein Anfang- und ein Enddatum. Die Angabe dieser Zeitspanne kann verschieden aufgebaut werden für das uGraph Framework wurde folgende Parametrisierung gewählt.

Parameter	Beschreibung
from	Anfrage mit „from" als Startdatum. Enddatum ist der aktuelle Zeitpunkt (als UTC Timestamp).
from, to	„from" als Startdatum und „to" als Enddatum.
from, ticksize	„from" als Startdatum, Enddatum ist der aktuelle Zeitpunkt. Die „ticksize" gibt den Abstand der Abgefragten Werte an.
from, to, ticksize	„from" als Startdatum, „to" als Enddatum. Die „ticksize" gibt den Abstand der Abgefragten Werte an.

Tabelle 6 Parameter für Zeitspannenangabe

8.3.2 Version 0.1b

In der Version 0.1b wurde die Abfrage vom uGraph Client zu uGraph Proxy als GET implementiert. Die Parameter wurden über die URL übergeben.

8.3.2.1 Parameterliste

Parameter	Beschreibung
DataAdapter Identifier	Dieser Parameter wird benötigt um zu spezifizieren welcher Time Series Service vom uGraph Proxy DataAdapter angesprochen werden soll.
Zeitangabe	Zeitspanne für welche die Time Series Daten abgefragt werden sollen.

Tabelle 7 Parameterliste Version 0.1b

8.3.2.2 Beispiel Abfrage

Nur über GET ansprechbare REST API.

```
GET /ugraph/rest/get/<data-adpater-id>/<from>

GET /ugraph/rest/get/<data-adpater-id>/<from>/<to>

GET /ugraph/rest/get/<data-adpater-id>/<from>/<ticksize>

GET /ugraph/rest/get/<data-adpater-id>/<from>/<to>/<ticksize>
```

Beispiel 1 Abfrage von Time Series Daten in der Version 0.1b

8.3.3 Verion 0.2b

Gegenüber der Version 0.1b wurde ein zusätzlicher Parameter (im folgenden Query Parameter genannt) für das Abfragen von einem bestimmten DataAdapter hinzugefügt.

Der Query Parameter kann für jeden Time Series Service anders aussehen. Aus diesem Grund wurde beschlossen dieser Parameter als JSON Objekt zu modellieren. So wird dem Entwickler die Freiheit überlassen beliebige Parameter für die Abfrage zu übergeben.

8.3.3.1 Parameterliste

Parameter	Beschreibung
DataAdapter Identifier	Dieser Parameter wird benötigt um zu spezifizieren welcher Time Series Service vom uGraph Proxy DataAdapter angesprochen werden soll.
Zeitangabe	Zeitspanne für welche die Time Series Daten abgefragt werden sollen.
Query Parameter	DataAdapter spezifische Parameter für das Abfragen eines konkreten Time Series Service.

Tabelle 8 Parameterliste Version 0.2b

8.3.3.2 Beispiel Abfrage

Nur über GET ansprechbare REST API. JSON als letzter Parameter

```
GET /ugraph/rest/get/<data-adpater-id>/<from>/{„queryparam“:„param1“}

GET /ugraph/rest/get/<data-adpater-id>/<from>/<to>/{„queryparam“:„param1“}

GET /ugraph/rest/get/<data-adpater-id>/<from>/<ticksize>/{„queryparam“:„param1“}

GET /ugraph/rest/get/<data-adpater-id>/<from>/<to>/<ticksize>/{„queryparam“:„param1“}
```

Beispiel 2 Abfrage von Time Series Daten in der Version 0.2b

8.3.4 Version 1

In dieser Version wurde zum Query Parameter, der Plugin Parameter hinzugefügt. Dieser dient der Parametrisierung der ProxyPlugins.

Weil die Datenstruktur des JSON durch den zusätzlichen Parameter immer komplexer wurde und um Fehler durch die Limitierung der URL-Länge zu vermeiden, wurde entschlossen aus der GET Anfrage eine POST zu machen. Aus diesem Grund wurden die Parameter nicht mehr über die URL übergeben sondern direkt über den Request Body in Form von JSON.

8.3.4.1 Parameterliste

Parameter	Beschreibung
DataAdapter Identifier	Dieser Parameter wird benötigt um zu spezifizieren welcher Time Series Service vom uGraph Proxy DataAdapter angesprochen werden soll.
Zeitangabe	Zeitspanne für welche die Time Series Daten abgefragt werden sollen.
Query Parameter	DataAdapter spezifische Parameter für das Abfragen eines konkreten Time Series Service.
Plugin Parameter	ProxyPlugin spezifische Parameter für das Aufrufen von ProxyPlugins.

Tabelle 9 Parameterliste Version 1

8.3.4.2 Beispiel Abfrage

Durch diese Änderung konnte die REST API auf eine einzige Funktion minimiert werden.

```
POST /rest/data/get-data

{
        "adapter"  : "<adapter-identifier>",
        "time" : {
                  "from" : "1400250000",
                  "to" : "14002500300",
                  "ticksize" : "1800",
                },
        "queryParams" : {"param1" : "example"},
        "plugins" : [{"pluginId" : "<plugin-identifier>",
                     "pluginParams" :
                        {
                              "param1" : "example"
                        }
                  }]
}
```

Beispiel 3 Abfrage von Time Series Daten in der Version 1

8.4 Evaluation des Datenformat für die Kommunikation zwischen Client und Proxy

Zu diesem Zeitpunkt der Bachelorarbeit war die Plotting Library bereits evaluiert. Da das Adaptieren der Time Series Daten auf dem uGraph Proxy verlegt wurde, musste das vom uGraph Proxy gelieferte Datenformat bereits von Dygraphs akzeptiert sein. Trotzdem wurden ein paar Ergänzungen am Datenformat vorgenommen um die Implementierung des uGraph Clients zu vereinfachen indem man den Zugriff auf die Daten erleichtert und diese in einer gewissen Struktur bringt.

Wie bereits in der Evaluation erwähnt (8.1.5.1 Dygraphs, Seite 27) unterstütz Dygraphs neben CSV als Datenformat auch JavaScript Arrays. Es wurde entschieden JavaScript Arrays für die Übergabe der Daten an die Plotting Library zu benützen, um so auch die Implementierung des uGraph Client einfach und verständlicher zu halten. Auch die Legende für Dygraps wird als JavaScript Array übergeben.

Um Legende und Time Series Daten in der Proxy Response unterscheiden zu können wurde JSON eingesetzt. Daraus folgte folgendes DatenFormat:

```
{
        "data" : [[],[]],
        "legend" : []
}
```

Beispiel 4 Übertragungsformat uGraph Proxy => uGraph Client

8.5 Architektur

Die Architektur wurde auf Wunsch des Auftraggebers IBM auf Englisch verfasst. Um den technischen Bericht einheitlich zu halten wurde die Architektur separat dokumentiert (siehe Solution Dokument im Anhang).

9 Resultat

9.1 Vergleich Vision und Resultat

Zu Anfang wurde eine Vision erarbeitet welche die erste Grundlage für das Projekt war und die erste Übersicht über die Features bot. Nun da das Resultat steht wird dieses mit der ursprünglichen Vision verglichen.

In der Vision wurde der Proxy als Server beschrieben, im Laufe des Projektes wurde jedoch klar, dass dieser viel mehr als Proxy funktioniert. Dies da er kein Endpunkt für eine Anfrage ist sondern Mittelsmann. Der Proxy empfängt die Anfrage und leitet sie mit Hilfe des entsprechenden DataAdapter an den Time Series Service weiter. Die Antwort des Time Series Server leitet er dann, aufbereitet, weiter an den Client.

Für das dynamische Aktualisieren der Daten im Graph wurde zu Anfang vorgeschlagen Push-Nachrichten vom Server zu Senden wenn neue Daten zur Verfügung stehen. Davon wurde im schlussendlichen Resultat abgesehen, da der Time Series Service dafür auch Push-Nachrichten unterstützen müsste, was nicht garantiert werden kann. Das Nachladen der Daten wurde über regelmässiges polling des Clients gelöst.

Die in die Vision beschriebenen Konzepte für das vereinfachen der Darstellung und Abfrage von Time Series Daten konnten mit dem uGraph Framework realisiert werden.

9.2 Umgesetzte Features

9.2.1 Zu Beginn Definiert

Dies sind die zu Anfang des Projektes mit dem Industriepartner und dem Betreuer vereinbarten Features.

Feature	Beschreibung
Time Series Diagramm zeichnen lassen	Der Entwickler ruft die Funktion des uGraph Frameworks auf um einen Graphen zu zeichnen. Dabei übergibt er die benötigten Parameter um den gewünschten Graph zu zeichnen.
Darstellungsoptionen setzen	Der Entwickler will das bestehende Diagramm ändern. Dafür verändert er die Darstellungsoptionen. Daraufhin wird das Diagramm vom uGraph Framework entsprechend angepasst.
ClientPlugin registrieren	Der Entwickler des Web UI möchte nach dem Abrufen der Time Series Daten weitere, kleinere Berechnungen auf den Daten durchführen. Dafür schreibt er ein ClientPlugin und registriert diesen auf dem uGraph Client
ProxyPlugin registrieren	Der Entwickler will die abgefragten Time Series Daten weiter für die Darstellung aufbereiten und registriert auf dem uGraph Proxy ein Plugin welches die gewünschten Berechnungen durchführt.
DataAdapter registrieren	Ein Entwickler will mit Hilfe des uGraph Frameworks ein neuer Time Series Service ansprechen, dafür registriert er ein DataAdapter auf dem uGraph Proxy.
Hover	Wenn mit der Maus über ein Datenpunkt im Graf gefahren wird, wird der Wert des Datenpunktes angezeigt.
Daten bei Schwenken nachladen	Wenn der Benutzer über die vorhandenen Daten im Diagramm hinaus pannt werden automatisch neue Daten nachgeladen.
Daten bei Zooming nachladen	Wenn der Benutzer im Diagramm zoomt werden ab einem konfigurierbaren zoom-level neue, detailliertere Daten nachgeladen. Beim herauszoomen werden wieder die vorherigen, weniger detaillierten Daten angezeigt.
Regelmässiges aktualisieren der Daten	Es kann ein Zeitintervall angegeben werden nach dem immer wieder neue Daten nachgeladen werden.

9.2.2 Im Verlauf des Projektes definiert

Feature	Beschreibung
Preloading	Es werden bei einer Abfrage bereits Daten vor und nach dem Anfangs- bzw. Endzeitpunkt abgefragt damit wenn der Benutzer pannt schon Daten angezeigt werden.

9.3 Weitere Ideen / Open Issues

Als weiter Features wären denkbar:

Feature	Beschreibung
Events im Graph markieren	Spezielle Ereignisse im Graph hervorheben.
Authentifizierung	Authentifizierung des Client beim Server damit nur Authentifizierte Clients mit dem Proxy kommunizieren können

Diese Features wurden nicht implementiert, sie den Rahmen des Projektes gesprengt hätten und neben den anderen Features nicht im Zeitrahmen der Bachelorarbeit realisierbar waren.

9.4 Auswertung Arbeitsaufwand

Für die Bachelorarbeit wurden insgesamt 730.50 Arbeitsstunden aufgewendet. Dies liegt im für eine Bachelorarbeit erwarteten Rahmen. Die Zeit ist gleichmässig auf beide Teammitglieder verteilt, 365.00 Stunden bei Simon Brouwer und 365.50 Stunden bei Giuseppe Aquino.

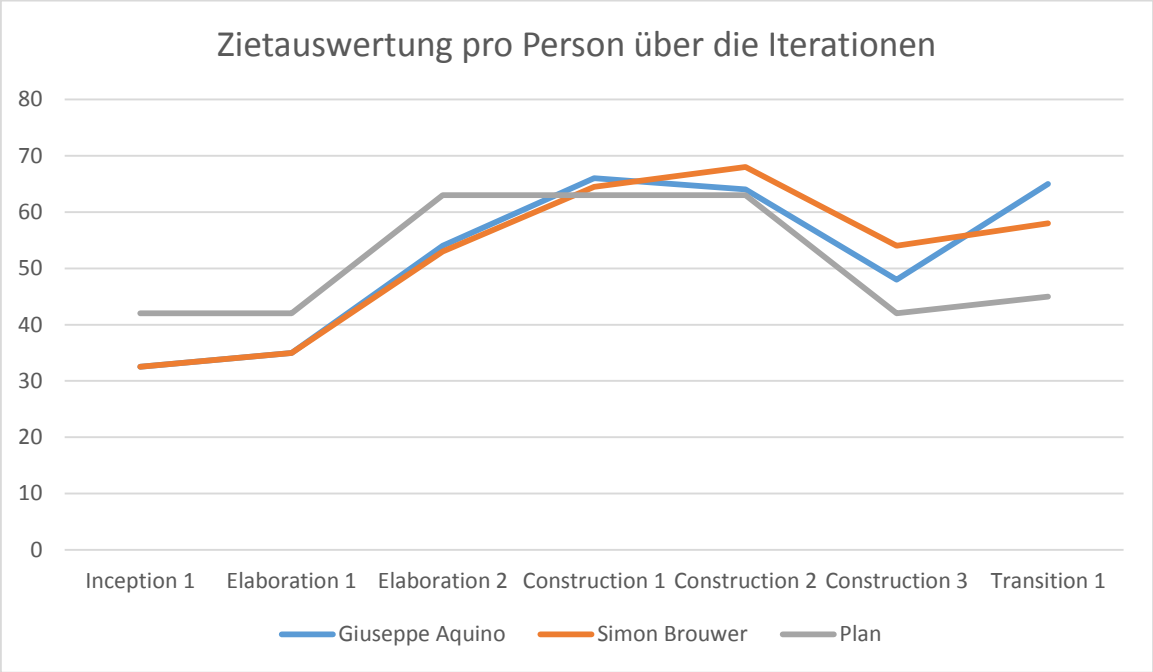


Abbildung 11 Diagramm Zeitauswertung pro Person über die Iterationen

Wie in Abbildung 11 ersichtlich stieg der Aufwand für das Projekt, wie erwartet, im Laufe des Projektes. Der Knick in der Construction 3 kann damit erklären das die Construction 3 nur 2 Wochen lang war im Vergleich zu Elaboration 2, Consturction 1 und 2 mit jeweils 3 Wochen. Die Transition 1 war auch 2 Wochen lang, der höhere Aufwand kann damit erklärt werden dass in dieser Iteration der Abschluss der Arbeit gemacht wurde und kein regulärer Unterricht mehr stattfand.

Abbildung 12 Diagramm Zeitaufteilung Aktivitäten

In Abbildung 12 kann erkannt werden, dass für die Dokumentation etwas mehr Zeit verwendet wurde als für die Implementierung. Dies ist darauf zurückzuführen, dass die Evaluation der Plotting Library und der Konzepte auch in der Dokumentation verbucht wurden. Dies würde in einem nächsten Projekt getrennt gebucht.

10 Quellen & Literaturverzeichnis

Abbreviation	Source
[WikJSPL]	Wikipedia. (11. Juni 2014). *Comparison of JavaScript charting frameworks*. Von Wikipedia: http://en.wikipedia.org/wiki/Comparison_of_JavaScript_charting_frameworks abgerufen
[DivJSPL]	Diverse. (11. June 2014). *JavaScript Chart Library*. Von Stackoverflow: http://stackoverflow.com/questions/119969/javascript-chart-library abgerufen
[FloLIB]	Flotr. (11. Juni 2014). *flotr*. Von flotr: https://code.google.com/p/flotr/ abgerufen
[SquLIB]	Square. (11. Juni 2014). *Crossfilter*. Von http://square.github.io/crossfilter/ abgerufen

11 Abbildungsverzeichnis

Time Series Service	Eine Datenbank für einen Web Service welcher eine Schnittstelle bietet um Zeitreihendaten abzufragen
Plotting Library	Bibliothek zur visualisierung von Zeitreihendaten in einem Graphen.
JNDI	Java Naming and Directory Interface
JAR	Java Archive
Dygraphs	Für das Framework verwendete Plotting Library
Proxy	Im Netzwerk: Vermittler der Anfragen Entgegennimmt und dann von seiner eigenen Adresse die Anfrage weiterleitet

13 Anhang

Projektplan

Giuseppe Aquino & Simon Brouwer

1 Inhalt

2 Einführung

2.1 Zweck

Dieses Dokument beinhaltet die detaillierte Beschreibung des Projektes. Es enthält die Aufteilung des Projektes in verschiedene Projektphasen und die festgelegten Meilensteine.

2.2 Referenzen

- Iterationsplanung

3 Projekt Übersicht

Im Rahmen der Bachelorarbeit soll ein Framework erstellt werden. Der Zweck ist es eine einheitliche und einfach zu bedienende API für die clientseitige Implementation des Web UI anzubieten um so die Visualisierung von Time Series Daten zu vereinfachen und die Performanz zu steigern.

3.1 Datenverwaltung

3.1.1 Code & Dokumentation

Die Codeverwaltung wird auf einem von der IBM gehosteten GIT Repository stattfinden.

Die Dokumentation für die Abgabe wird, sofern diese keine kritische Daten der IBM enthält, auf dem für die Bachelorarbeit eingerichteten Google Drive gespeichert.

3.1.2 Testdaten

Die während der Arbeit verwendeten Testdaten werden direkt aus zwei verschiedenen IBM Internen Projekten bezogen, dabei handelt es sich um einen Systemanalyse Tool und ein Netzwerkanalyse Tool.

Beide Projekte verfügen über eine API für den zugriff auf die Time Series Daten, weshalb die Testdaten nie auf einem Lokalen PC zu liegen kommen, sollte das trotzdem der Fall sein, sind die Daten vertraulich zu behandeln.

3.2 Richtlinien für Code

3.2.1 Codestyling

Um den Code einheitlich zu halten werden Styletemplates eingerichtet und direkt in der Entwicklungsumgebung eingebunden.

3.3 Lieferumfang

- Implementation Library (inkl. Unit-Tests)
- Beispiel Plugins für die beiden Referenzprojekte
- API Dokumentation (mit Benutzungsbeispielen)

3.4 Annahmen und Einschränkungen

Das Projekt wird mit Daten aus zwei IBM Projekten getestet. Die finale Implementation soll generisch gehalten werden damit sie für spätere Projekte einsetzbar ist. Für allfällige Anpassungen an spätere Projekte, wird die Unterstützung von Plugins geplant, welche die Anpassung ermöglichen sollen.

4 Projektorganisation

4.1 Team

4.1.1 Giuseppe Aquino

Kenntnisse in: Java, C++, PHP, SQL, JavaScript

Motivation: Das Arbeiten mit Webtechnologien war schon immer ein grosses Hobby von mir. Nach der Studienarbeit bei IBM war mein Interesse an Webtechnologien noch grösser und ich fühlte mich bereit eine grössere Anforderung anzunehmen.

4.1.2 Simon Brouwer

Kenntnisse in: Java, C++, PHP, SQL, JavaScript

Motivation: Webentwicklung war schon seit Anfang meiner Ausbildung ein Hobby von mir, welches ich in meiner Freizeit und später bei der Arbeit, ausgeübt habe. Deshalb freue ich mich umso mehr nun auch für die Bachelorarbeit mit einem grossen Industriepartner wie IBM zusammenzuarbeiten und meine Kenntnisse zu Vertiefen.

4.2 Externe Schnittstellen

Betreuer: Prof. Dr. Markus Stolze

Externe Projektbetreuer: Dr. Marc Stöcklin
Dr. Mike Osborne

5 Management Abläufe

Für das Projekt steht ein Semester à 17 Wochen zur Verfügung. Bis zur Deadline werden folgende Deliverables erwartet:

- Projektdefinition
- Anforderungsanalyse
- Architektur Dokumentation
- Lauffähiger Code entsprechend der Anforderungsanalyse und der Architektur-Dokumentation
- Dokumentation zur API & Benutzungsbeispiele
- Abschlussbericht und Abstract sowie weitere Dokumentationen entsprechend der Anforderungen der Abteilung Informatik der HSR

5.1 Grobe Iterationsplanung

Das Projekt wird mit RUP geführt, weshalb die Iterationsplanung jeweils vor Beginn der Iteration gemacht wird. Für eine Übersicht der Arbeitspakete wird auf der Iterationsplanung verwiesen.

5.1.1 Phasen / Iterationen

5.1.1.1 Inception 1 (17.02.2014 - 02.03.2014)

Während der Inception Phase wird mit dem Partner IBM die Vision diskutiert und allfälligen Fragen geklärt. Anhand der Vision wird das Visionsdokument erstellt.

In einem Meeting mit den Teammitgliedern der beiden IBM Projekte sollen die Nutzungsszenarien für das Framework festgelegt werden.

Das Team beginnt mit der Erarbeitung der Anforderungen an das Framework. Ausserdem wird die Entwicklungsumgebung eingerichtet, sowie verschiedene Management Tools.

5.1.1.2 Elaboration 1 (03.03.2014 - 16.03.2014)

In der ersten Elaborationsphase sollen die Anforderungen genauer ausgearbeitet und verfeinert.

Ausserdem wird eine Risikoanalyse durchgeführt um die möglichen Risiken zu erkennen, einzugrenzen und eventuell schon entfernen zu können.

In dieser Phase werden Open Source Plotting-Libraries für die Verwendung im Projekt evaluiert. Dementsprechend wird die Dokumentation der Evaluation erstellt und vom Industriepartner IBM abgenommen.

5.1.1.3 Elaboration 2 (17.03.2014 - 06.04.2014)

In der zweiten Elaborationsphase wird ein Prototyp des Frameworks entwickelt. Dabei wird der Fokus hauptsächlich auf die Architektur des Frameworks sein.

Die Plotting Library sollte zu diesem Zeitpunkt bereits bekannt sein und es können bereits Abklärungen bezüglich einzusetzende Frameworks und Libraries gemacht werden für die Unterstützung der Entwicklung.

Die Architektur wird mit einem Assistenten der HSR und mit der IBM besprochen. Aufbauend auf dem ersten Architekturentwurf werden die ersten Prototypen und Proof of Concepts des Frameworks erstellt.

5.1.1.4 Construction 1 (07.04.2014 - 27.04.2014)

Während der Construction I Phase wird mit der Implementierung der Lösung begonnen. In dieser Phase werden bereits Code-Reviews vorgenommen um eine saubere Grundlage für die weiteren Phasen zu garantieren.

Der Fokus dieser Phase wird auf die Implementierung des Proxys und des Client Seite sein und deren Kommunikation. Dabei sollte stets an die Ausbaufähigkeit gedacht werden.

5.1.1.5 Construction 2 (28.04.2014 - 18.05.2014)

Nach der Construction I sollte die Kommunikation zwischen Proxy und Client vorhanden sein. Der uGraph Client soll mit der Plotting Library arbeiten können.

Der nächste Schritt liegt darin die Erweiterbarkeit des Frameworks zu auszubauen. Die erste ProxyPlugins oder gar DataAdapter werden bereits während dieser Phase geschrieben.

5.1.1.6 Construction 3 (19.05.2014 - 01.06.2014)

Die Proxyseitige Plugin-Infrastruktur sollte soweit stehen und es kann mit dem Implementieren der DataAdapter für die beiden IBM Projekte begonnen werden.

5.1.1.7 Transition 1 (02.06.2014 - 13.06.2014)

In der Abschlussphase des Projektes wird das definitive Release erstellt und ein letztes Mal getestet. In dieser Phase werden keine neuen Features mehr implementiert, stattdessen wird die API Dokumentation geschrieben. Sämtliche Dokumente für die Abgabe werden überprüft und fertiggestellt.

5.1.2 Meilensteine

Meilenstein	Resultate	Termin
MS1 End of Inception	Projektplan, Visionsdokument	02.03.2014
MS2 End of Evaluation für Plotting Library	Für das Projekt einsetzbare Plotting Library	16.03.2014
MS3 Ende Elaboration	Abgenommene Architekturprototyp	07.04.2014
MS4 Erster DataAdapter	Erster DataAdapter ist implementiert und lauffähig.	19.05.2014
MS5 Work Done	Final Release API Dokumentation (Benutzerhandbuch)	13.06.2014

5.2 Besprechungen und Meetings

Meetings werden wie in den folgenden Punkten behandelt. Die Protokolle der Sitzungen sind Bestandteil der Arbeit und werden im Administrativen Anhang mit abgegeben.

5.2.1 HSR Meetings

Meetings mit dem Betreuenden Dozent werden wöchentlich gehalten. Die Sitzungen werden Protokolliert. Die Protokolle sind auf Wunsch auch für die IBM ersichtlich.

5.2.2 IBM Meetings

Meetings mit dem Industriepartner IBM werden wenn möglich wöchentlich gehalten. Protokolle müssen vom verantwortlichen Betreuer der IBM abgenommen werden.

6 Infrastruktur

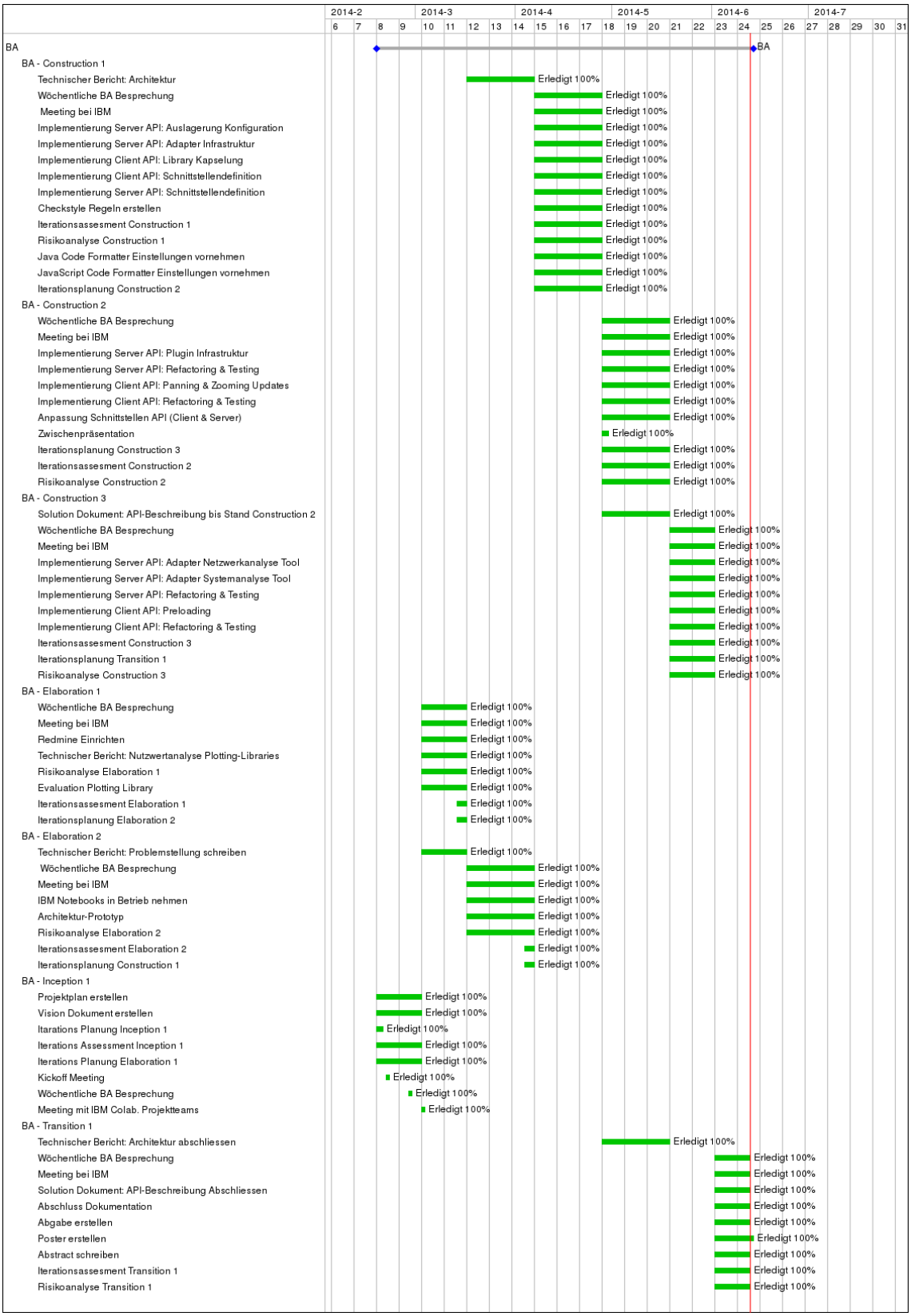
Das gemeinsame Arbeiten findet in den von der HSR zur Verfügung gestellte Räume statt und bei notwendiger Anwesenheit bei der IBM ZRL in Rüschlikon. Die Implementationsarbeit wird dabei auf den von der IBM zur Verfügung gestellten Rechnern stattfinden, während für die Dokumentation die Rechnerwahl offen ist. Die Verwaltung der Arbeitspakete und die Zeiterfassung wird im Redmine erfolgen welches auf einem virtuellen Server der HSR installiert ist. Die Versionsverwaltung wird auf Wunsch der IBM auf einem GIT Repository, der sich auf einem IBM Server befindet, geregelt.

6.1 Software

Während dem Projekt wird mit folgender Software gearbeitet:

- IDE (Eclipse für Web und Java)
- Redmine (Zeiterfassung)
- GIT
- MS Office (für die Endverarbeitung)

7 Arbeitspakete



Vision

Giuseppe Aquino & Simon Brouwer

1 Contents

2 Einführung

Dieses Dokument dient als erste Analyse und Definition der Anforderungen an das das zu entwickelnde Framework für die Verarbeitung und Visualisierung von Timeseries.

Unter Timeseries wird eine Folge von zeitabhängigen Messwerte oder Beobachtungen verstanden welche den Benutzer bei der Analyse eines Systems unterstützen. Typische Anwendungsbereiche für Timeseries sind z.B. die Entwicklung von Aktienkurse, Preise oder Computer- bzw. Netzwerkperformanz. Das Problem dass sich dabei ergibt ist die Visualisierung von Timeseries auf dem Userinterface. Bei der grossen Vielzahl an Anwendugen, welche zur Auswertung Timeseries einsetzen, ist es immer schwieriger eine einheitliche UI Implementation anzubieten. Ziel des Projektes ist es einen Framework zu kreieren, welches dem Entwickler eine API anbietet um die Userinterface Programmierung mit möglichst wenigen Anpassungen zu vollenden.

2.1 Scope (Brief)

Im Rahmen der Bachelorarbeit soll für den Industriepartner IBM einen Framework erstellt werden, welches erlaubt die Darstellung von Timeseries im Webbrowser zu vereinfachen. Das Framework wird während der Bachelorarbeit, mit zwei IBM Projekte, IBM Smart Grid und IBM ZiMon, getestet weshalb die Implementation stark auf diese zwei optimiert wird. Die Projekte dienen gleichzeitig als Lieferanten von Testdaten.

2.2 Definition, Akronyme und Abkürzungen

Definition	**Erklärung**
Client API	Clientseitige Schnittstelle des geplanten Systems. Kommuniziert mit der Server API und dem Userinterface welches eine separate Library für die Visualisierung der Timeseries Daten implementiert.
Server API	Serverseitige Schnittstelle des geplanten Systems. Kommuniziert mit der Client API und dem Datenlieferant (Datenbank, Webservice)
JSON	Datenformat zur Strukturierung eines Programmobjektes.
XML	Datenformat zur Strukturierung eines Programmobjektes.
Server-Push	Server benachrichtigt Client bei Änderungen. (Klassisch: Client holt Daten vom Server)
Adapter Framework	Das zu entwickelnde System. Adapter genannt weil es zur Adaptierung, bzw. Anpassung der Daten/Datenformate an den Bedürfnisse des Userinterfaces dient.
Plotting Library	Library welche benutzt wird um die Timeseriesdaten graphisch darzustellen.

2.3 References

- Appendix A (Statement of Work of Agreement No. 2014_017)
- Arbeitsbeschreibung "Visualisierung von Time Series Daten mit HTML 5"
- Protokoll Kick-Off Meeting

3 Anforderungen

3.1 Problembeschreibung

3.1.1 Problem 1

Das Problem Visualisieren von Timeseries

betreffend	Entwickler des Userinterfaces
bewirkt	Eine schwerfällige und mit Aufwand verbundene UI Entwicklung.
Das neue System würde anbieten	• Einfachere Integration ins UI via Client API • Performanzsteigerung in der UI Steuerung durch verbesserte Handhabung der Timeseriesdaten (Caching, verbessertes Eventhandling) • Dynamische Aktualisierung des UI via Server-Push-Notifizierung.

3.1.2 Problem 2

Das Problem Antworten von verschiedenen Datenquellen (Datenbanken, Webservices) mit verschiedener Datenstruktur (JSON, XML, etc.) verarbeiten können

betreffend	Entwickler des Userinterfaces
bewirkt	Mühsames parsen und adaptieren von Daten für die Darstellung auf dem Userinterface.

Das neue System würde anbieten	• Standardisiertes Antwortformat realisiert durch Server API
	• Klare Trennung von Code (Server/Client)
	  ○ Client: Zuständig für die Visualisierung der Timeseries Daten und für Usereventhandling.
	  ○ Server: Zuständig für die Aufbereitung der Daten im entsprechenden Format

3.2 Einschränkungen

Das Adapter Framework das zu Entwickeln gilt soll möglichst ohne Opensource Librarys auskommen. Sollte trotzdem der Einsatz von Opensource Librarys notwendig sein so muss das zuerst mit der IBM besprochen werden. Opensource Komponente die keine Zulassung von der IBM erhalten haben, müssen vermieden werden.

4 Stakeholder

Das Framework wird geschrieben um die Arbeit in der Entwicklung der Visualisierung von Timeseries Daten zu erleichtern. Von der Optimierung profitieren in erster Linie die Entwickler welche sich mit der Programmierung von UI's befassen. Aufgrund verschiedenartigen Timeseriesdaten, mit unterschiedlichen Datenformate und Zeitformate (Zeit, einfache Dezimalzahlen etc.) ist die Darstellung oft statisch realisiert, z.B. muss das holen der Daten für jeden Graphen anders implementiert werden. Im Bezug auf Datenformate sind auch die Antworten die von den Datenquellen geliefert werden zu erwähnen, welche oft unterschiedliche Datenstrukturen als Antwort liefern. Ein typisches Beispiel dafür ist der Einsatz von JSON oder XML als Antwortformat.

4.1 Stakeholder Summary

Name	Beschreibung	Verantwortlichkeiten
Entwickler	Entwickelt das Userinterface. Möchte eine möglichst einfache Implementation der Timeseries.	• Entwickeln des Userinterface

5 Vision

5.1 Plotting Library

Bevor mit der Implementation der Client/Server API gestartet werden kann, muss eine Plotting Library gewählt werden. Dafür werden während einer Evaluationsphase verschiedene Plotting Libraries auf Performance und vor allem auf Funktionalität getestet. Das System wird dann auf diese Library aufbauen.

5.2 Client/Server API

Es wird ein Adapter Framework geplant um die obengenannten Probleme zu lösen. In einer ersten Diskussion mit dem Auftraggeber wurden die folgenden Features als nötig und sinnvoll erkannt. Wobei die Angabe der Risiken noch sehr rudimentär ist und nach einer späteren Analyse in einem Separaten Dokument weitergeführt wird. Folgend eine Liste der möglichen Funktionalität des Framework.

Feature/Analyse	**Beschreibung**	**Priorität**	**Risiko**
1.0 Zusammenarbeit mit Plotting Library	Client API ist in der Lage mit der Plotting Library zu arbeiten.	1	Mittel
1.1 Server-Push Untersützung	Durch das Adapter Framework ist es möglich, dass Datenänderungen vom Server via Push auf die Plotting Library angezeigt werden.	1	Mittel-Hoch
1.2 Optimieren der Zooming Funktion der Plotting Library	Die Zoom Funktion der Plotting Library soll durch die Client API optimiert werden.	1	Mittel
1.3 Unterstützung verschiedener Datenformate der Timeseries	Client API soll in der Lage sein verschiedene Datenformate wie z.B. Zeit, Dezimalzahlen zu erkennen und der Plotting Library mitzuteilen.	1	Tief
1.4 Plugin-Unterstützung	Die Client API unterstützt die Anwendung von Plugins.	1	Mittel
2.0 Aufbereitung der Daten auf dem Server	Umwandlung der Daten welche von der Datenquelle geliefert werden in einem von der Plotting Library unterstützten Format.	1	Tief
2.1 Plugin-Unterstützung	Das Serverseitige API unterstützt die Anwendung von Plugins.	2	Mittel

max. Prio = 1; min Prio = 3;

Legende

- 1.x Betreffen Client API
- 2.x Betreffen Server API

6 Environment

6.1 IDE

- Eclipse (http://eclipse.org/)

6.2 Programmiersprache & Codequalität

- JavaScript
  - jQuery (https://jquery.org/)
- Java (http://www.java.com/)
- Checkstyle (http://checkstyle.sourceforge.net/)

6.3 Server

- Tomcat (http://tomcat.apache.org)

Solution

Giuseppe Aquino & Simon Brouwer

1 Contents

1. Introduction

1.1 Purpose

This document gives a detailed description of the uGraph framework and its architecture.

1.2 Overview

The uGraph framework is a framework to help developers visualize their Time Series data. The framework is split into two components the server side, which consists of a proxy and the client side. Both components are needed for working with uGraph. The client is mainly for the visualization of the data while the responsibility of the proxy consists in requesting the data from any desired Time Series Service and converting it for the direct use on the client.

1.3 Goals

The main goals of the framework are:

- Easy exposure of time series data sources
- Easy visualization of time series data in web pages
- Achieve flexibility with the usage of adapters for time series data sources

Beside those features, the framework should offer an API for the client development which includes the following functions:

- Dynamic pre-loading of data
- Dynamic loading of new data (after pan or zoom on the graph)
- Setting up callback-functions

The Proxy offers a REST Service for calling the data. It should be extendable, offering the developer a way to add more time series services easily.

In Figure 1 can be seen how the data flow looks with the current implementations of such systems and how it should change with the use of uGraph.
Currently the client makes a request to a web service on a server that stores the time series data. To do so the client has to know the address of the server and how the web service of that server works. The server responds to the request by sending the requested time series data in a format that the server dictates. The client receives the data and has to convert the data to a format that the plotting library it is using can interpret it and plot the data.
With the new architecture the client does not have to know the address of the server with the time series data and does not have to convert the data to be able to plot it. This I achieved by introducing the uGraph Proxy and a uGraph Client that wraps a plotting library. In the new system the client makes a request to the uGraph proxy for time series data. In that request it specifies an identifier for an adapter. The server then loads the adapter with the help of the identifier. The adapter is specially built to communicate with a specific time series data. The adapter then loads the time series data from the time series server and returns it to the uGraph proxy which

converts the data into a format that the client can understand and sends it to the client. The client then plots the data without having to convert it. This relieves the client form a lot of work and runs complex calculations on a powerful proxy.

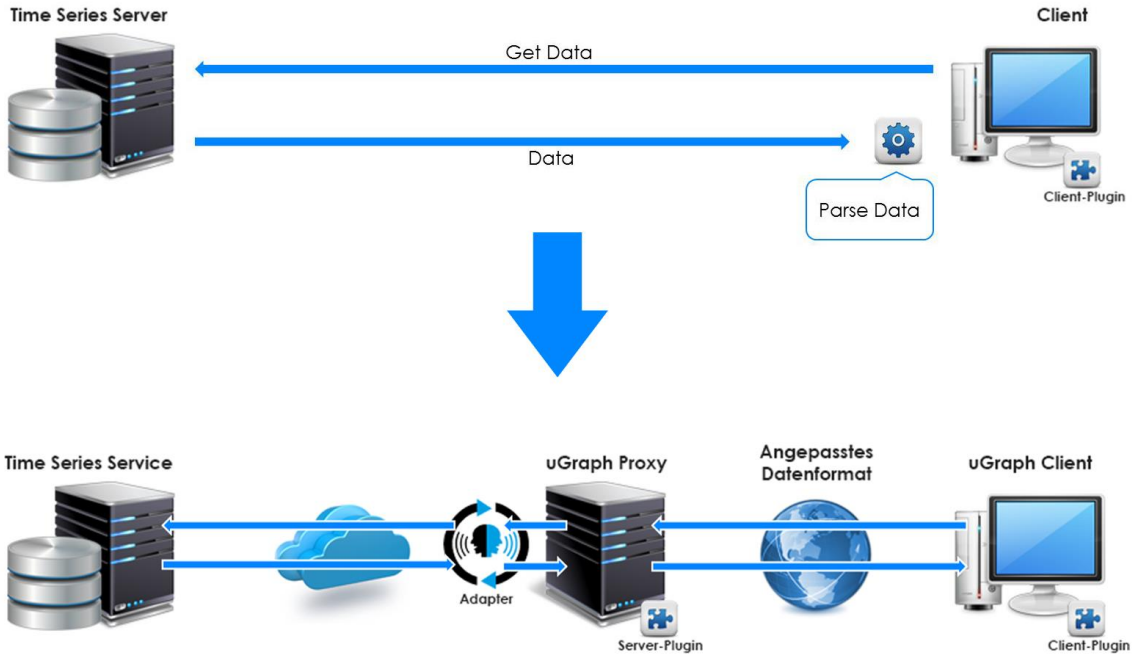


Figure 1 Actual state vs. desired state

1.4 Restrictions

The following restrictions were set up for the project by the bachelor team.

1.4.1 uGraph Client

- The client should be written in JavaScript
- All the used Libraries should be Open Source
- The license of the Plotting Library has to be accepted by IBM
- Support for Firefox 24.5.0 ESR

1.4.2 uGraph Proxy

- TomCat 7 as web container

1.5 References

- uGraph Proxy API
- uGraph Client API

2 Framework overview

uGraph is split into 3 tiers: the client for the visualization of the data which runs in the browser of the client, the proxy which runs a web service and is responsible for the parsing and adapting of the data and the time series service. The third tier is an already existing system. This last tier is not part of the implementation but is still needed to get data.

The proxy doesn't own any data but gets data from a time series service or database. The uGraph Client & the uGraph Proxy are the parts of the framework which were implemented during the bachelor thesis.

Dygraphs, which represents the plotting library used to realize uGraph, was evaluated during the bachelor thesis using IBM criteria.

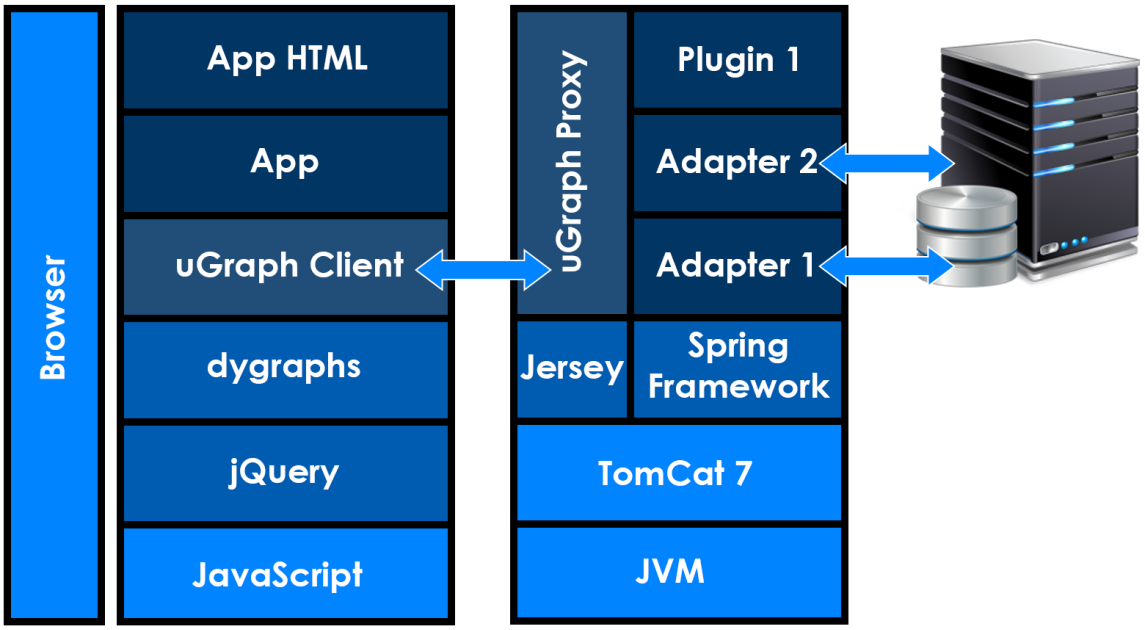


Figure 2 Technology stack uGraph

2.1 Components

Following a brief description of each component of the framework. A more detailed description can be found under the section Logical Architecture (Client / Proxy).

2.1.1 uGraph Client

The uGraph Client is responsible for the visualization of the called data. The client is based on the plotting library dygraphs, which means it uses the dygraphs functions to visualize the data from the proxy. The client is written in JavaScript with jQuery as additional Framework, which is also needed to run dygraphs.

2.1.2 uGraph Proxy

The uGraph Proxy is written in Java. For the communication with the client, a Jersey REST web service has been set on top of the proxy, which also means it has to be deployed in a web container. In the case of uGraph TomCat 7 was used as web container because of the requirements listed above. For the extensibility of the proxy, the Spring Framework has been used. The Spring Framework is mainly used for the dynamic loading of classes and realizes the ProxyPlugin & DataAdapter infrastructure.

2.1.2.1 Adapter

An adapter is located on the proxy. It is responsible for loading requested data from a specific time series data server and converting it into a format that the proxy can understand. When the proxy get a request for time series data it extracts the adapter identifier from the request and loads the corresponding adapter. The adapter then makes a request to a time series data server and converts the data before returning it to the proxy.

2.1.2.2 Plugin

A plugin is located on the proxy. A plugin is always run after the adapter has fetched the data. It gets loaded just like an adapter but does not load new data from a server but works with the data that the adapter loaded. Plugins are meant for making complex calculations on the data loaded by the adapter to prevent costly calculations on the client. Multiple plugins can be executed on the server. Each plugin will be executed after the other and has to work with the data that might have been modified by the previous plugin.

2.2 Example of a request

Following example shows the sequence of a possible request. The function names are simplified and are only meant for understanding the procedure.

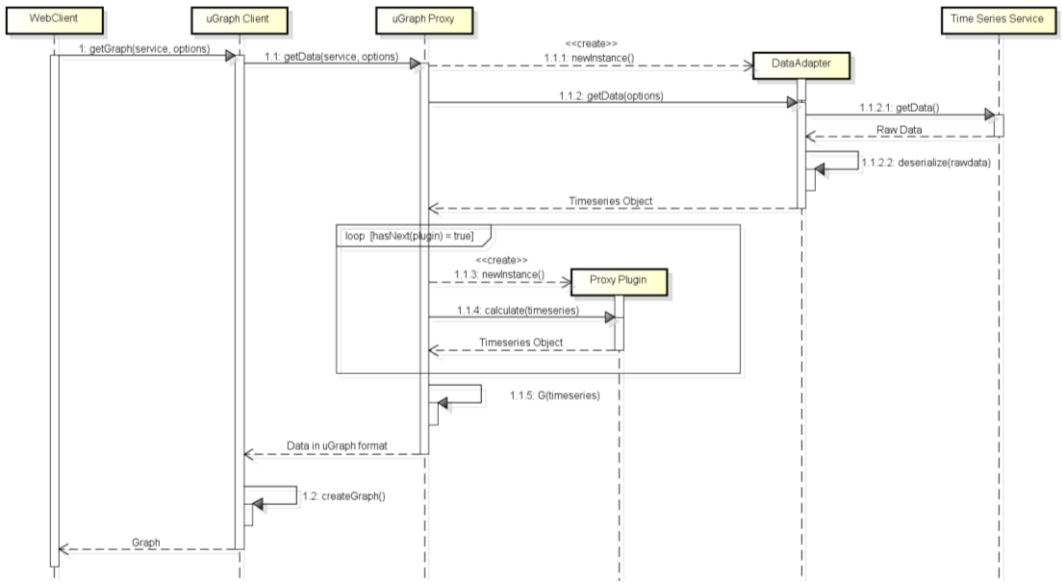


Figure 3 Example of a possible request

3 Logical Architecture Client

On the client side the Revealing Module Pattern[1] is used to structure the application. This pattern allows creating a base namespace and adding different modules to it. It also enables privacy, state and organization through closures. The Pattern was chosen to be able to structure the application and get a code style that is close to Java. It also allows easy extensibility of the application by adding or replacing a module. A module is JavaScript a function which returns only those module members that are meant to be public. Following a short example for understanding the principle:

```
var myRevealingModule = function() {
	var publicString = "this is accessable from outside";
	var privateString = "this is not accessable from outside";
	function privateFunction() {
		console.log("private function called");
	}
	function publicFunction() {
		privateFunction();
	}
	// Exposes only public functions and variables
	return {
		runPublicFunction : publicFunction,
		publicString : publicString
	};
};
var myObject = new myRevealingModule();
myObject.runPublicFunction();
```

With help of the Revealing Module Pattern the Relaxed Layers Pattern[2] is implemented to give the application a clear structure. The Revealing Module Pattern is responsible for the definition of the single modules which represent the layers. Also the Relaxed Layers Pattern separates the application into logical layers, which are stacked on top of each other. A top layer can access any lower layer but a lower layer cannot access layers above it.

As in Figure 4 can be seen, the application is separated into three layers, the presentation-, business logic- and data acces-layer. The presentation layer can access the business logic and the data access layer, as marked with the green arrows. The business logic layer can access the data access layer but not the presentation layer, as illustrated with the red and green arrow.
The Relaxed Layers Pattern was used instead of the Strict Layers Pattern because the presentation layer often has to access the data access layer, with the Strict Layers Pattern it would always have to go through the business logic layer, which would not add any functionality but just forward the call to the data access layer. So in order to prevent another layer of indirection and make sure the business layer does not get bigger than necessary the Relaxed Layers Pattern was used.

[1] [AddRMP]
[2] [BusRLP]

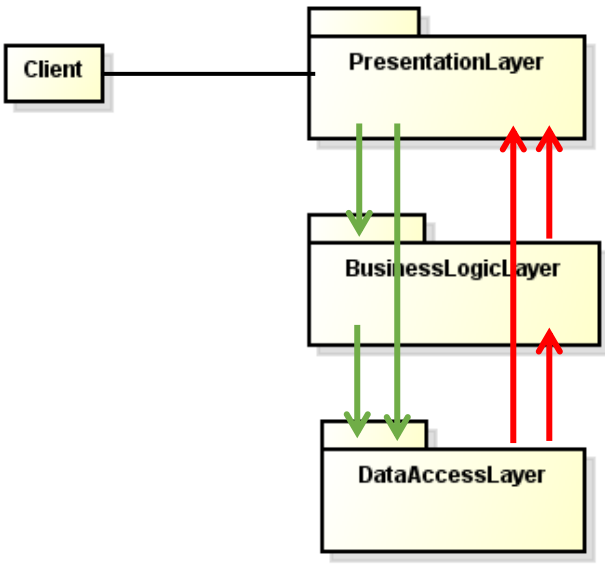


Figure 4 Example of Relaxed Layers Pattern

3.1 Namespace uGraph

uGraph is the base namespace for the application. It contains all the modules. It also includes JQuery, window and document to the scope to make sure their functionality can always be accessed inside the namespace and prevent conflicts with other libraries that might be used by the uGraph library's user.

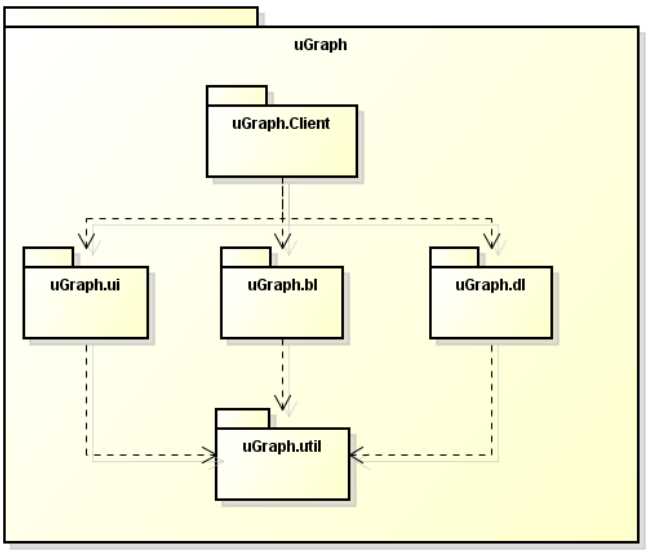


Figure 5 Overview of the Namespace uGraph

3.2 Module uGraph.util

The uGraph.util module provides utilities, such as checking if the provided time for a request is valid or checking that the provided DOM Element to draw the graph in, actually is a valid DOM Element. It initializes itself and does not have any dependencies on the other modules.

3.3 Functions

checkIsString(string):
Checks if the provided parameter is a string.

checkTimeCombination(time):
Checks if a valid time object was provided. A time objet is valid if one of the following combinations is given:

- From
- From, to
- From, ticksize
- From, to, ticksize

checkDivParams(graphElement, legendElement):
Checks if the provided parameters are valid DOM Elements. This is to make sure dygraphs can plot the graph in the given DOM Elements.

isNumber(number):
Checks if the provided parameter is a valid number. The !NaN() is not sufficient because values like ‘100’ or “100” pass the check.

checkIsFunction(func):
Checks if the parameter is a JavaScritp function.

checkPlottingOptions(options):
Checks if the options for the uGraph.ui are valid. This helps to prevent from loading new data too often or loading the wrong data.

3.4 Module uGraph.dl

The uGraph.dl module represents the Data Layer. It stores all the time series data and the data transfer objects needed for the communication with the server, which are described below. It also contains the functionality to request data from the proxy and works, in this case, also as communication layer.

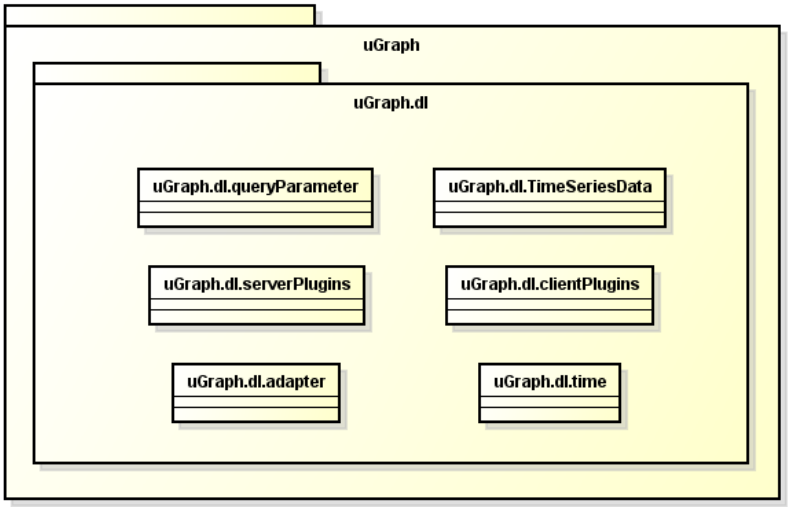


Figure 6 Overview uGraph.dl module

3.4.1 uGraph.dl.time

The time module maintains the time related data. It is a data transfer object that contains the time declaration needed for the request.

3.4.2 uGraph.dl.adapter

The adapter module maintains the adapter identifier which specifies which adapter should be called on the server.

3.4.3 uGraph.dl.queryParameter

The queryParameter module contains the parameters needed for a request. These parameters may vary from adapter to adapter. A missing parameter may cause a wrong answer or even an error. The needed parameters are typically defined by the DataAdapter developer.

A detailed description of the provided functions can be found in the Client-API.pdf

3.4.4 uGraph.dl.ProxyPlugins

The ProxyPlugins module maintains a list with plugins that should be executed on the proxy.

A detailed description of the provided functions can be found in the Client-API.pdf

3.4.5 uGraph.dl.clientPlugins

The clientPlugins module maintains the client side plugins. It offers functionality to register and deregister client side plugins. Registered plugins can then be set to be executed on the time series data.

A detailed description of the provided functions can be found in the Client-API.pdf

3.4.6 uGraph.dl.timeSeriesData

The timeSeriesData module maintains the time series data returned from the server. The data is stored in the same format as it is received form the proxy, because dygraphs uses the format and so the data does not have to be parsed in to another format to be plotted. The timeSeriesData module also provides functionality for manipulating the stored data.

A detailed description of how the data format looks can be found in the Client-API.pdf

3.4.6.1 Functions

getData()
Returns the time series data and the legend.

get/setSeriesData():
Returns/Sets the time series data.

get/setSeriesLegend():
Returns/Sets the legend entries.

addValueAtIndex():
Adds a value to the time series data at the specified index in the array.

getValuesAtIndex():
Returns the values from the time series data at the specified index.

3.5 Module uGraph.bl

The uGaph.bl module represents the Business Logic Layer. It handles the execution of the client side plugins. It also maintains an abstraction of the time series data that allows easier access and manipulation of the data for the plugins.

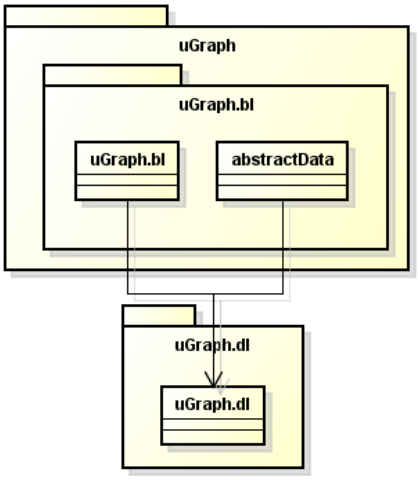


Figure 7 Overview uGraph.bl module

3.5.1 uGraph.bl.abstractTimeSeriesData

The abstracTimeSeriesData is an abstraction of the time series data stored in the uGraph.dl.timeSeriesData. It is meant to make it easier for a plugin developer to manipulate the time series data.

A detailed description of the abstractTimeSeriesData can be found in the Client-API.pdf document.

3.5.2 Important internal processes

The uGraph.bl is called after data was loaded from the server. The uGraph.bl then checks if there are any client side plugins that have to be run. If there are plugins to run the abstractTimeSeriesData object is initialized. Then the plugins are run sequentially after each other. Each plugin is passed the updated data from the previous plugin. This is illustrated in the sequence diagram below.

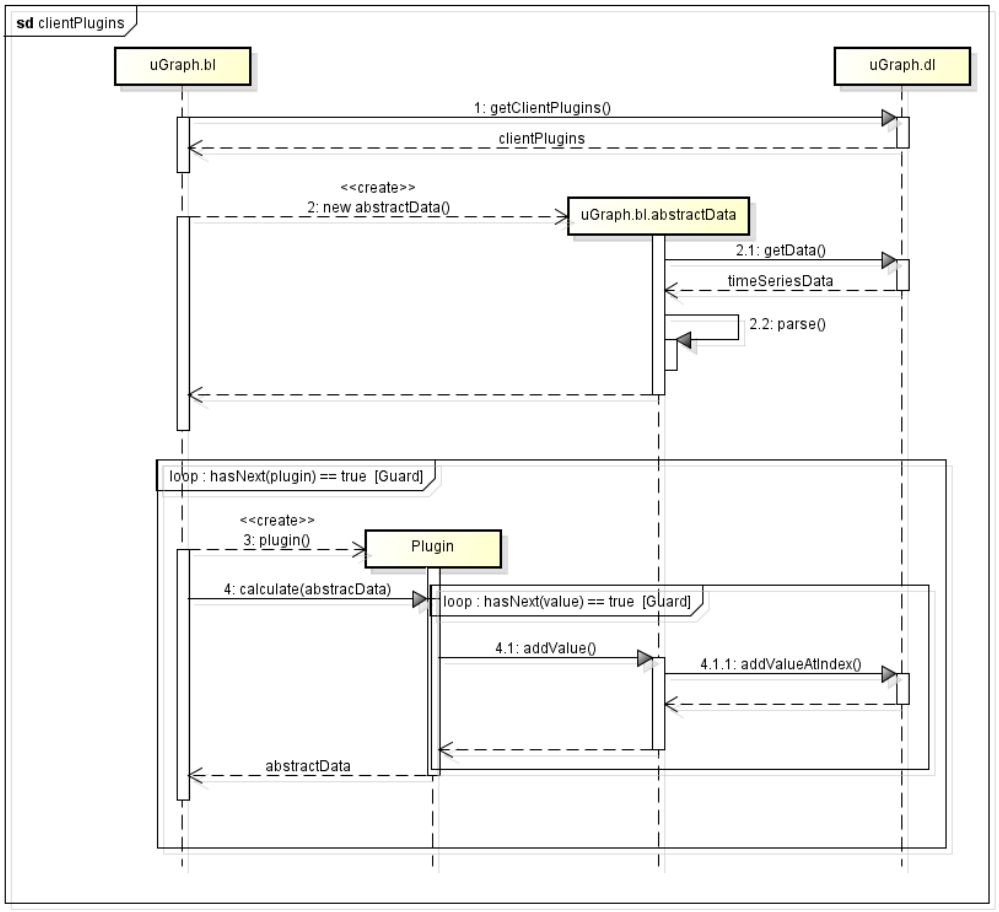


Figure 8 Sequencediagramm - Execution of ClientPlugin

3.6 Module uGraph.ui.graph

The uGraph.ui.graph module represents the User Interface Layer. It contains all the functionality to draw, alter and update time series graphs. The module is a wrapper of the dygraphs plotting library and adds new functionality to it. It stores the settings and states of the graph and offers functions to access and update these. It also handles the user-interaction with the graph.

The settings (options) are specified in the Client-API.pdf.

Possible interactions are:

- Registering events for zooming and panning inside the graph.
  - Loading more data if needed
  - Reloading data
- Updating the graph with new data
- Redrawing the graph with different settings (options)

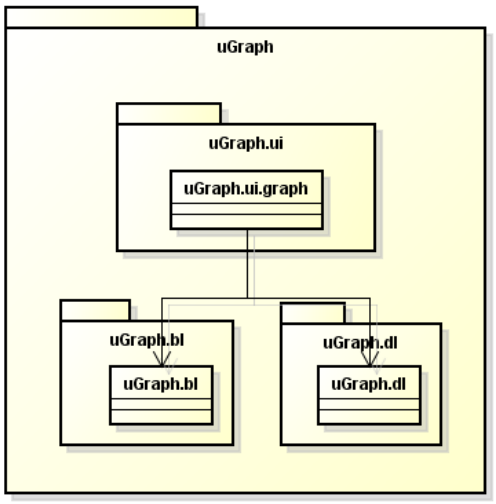


Figure 9 Overview uGraph.ui module

3.7 Module uGraph.Client

The uGraph.Client is the "main" module. It sets up the environment and allows access to the other modules. It initializes the Data Layer- (uGraph.dl), Business Layer- (uGraph.bl) and User Interface Layer- (uGraph.ui). The user accesses the framework functionalities through the uGraph.Client module which then forwards the call to the corresponding module.

A detailed description of the uGraph.Client functions can be found in the Client-API.pdf.

3.8 Architectural decisions

3.8.1 Use of the JQuery library

3.8.1.1 The Problem

Add cross browser support quickly without the need to renounce on JavaScript.

3.8.1.2 Constraints

- JavaScript as programming language
- Works with dygraphs plotting library

3.8.1.3 Treats

- Foreach functionality
- Ajax calls
- DOM manipulation

3.8.1.4 Decision

Because dygraphs requires the JQuery library to work JQuery was set to be used for the project. Although prototype.js also is a good library the team had already worked extensively with JQuery and it would just add another library that has very similar functionality like JQuery that would have to be used anyway.

4 Logical Architecture Proxy

Taking a look on the whole environment including the time series service we have a 3 Tier infrastructure. One tier for the client, one for the middleware which is in that case the uGraph Proxy and the time series services on the third tier.

The proxy is the middleware implementing the integration logic. It owns an API to access the time series services that represent the server implementing the application logic and resource layer that allows the access on the data.

On Figure 10 an overview of the single packages of the uGraph proxy.

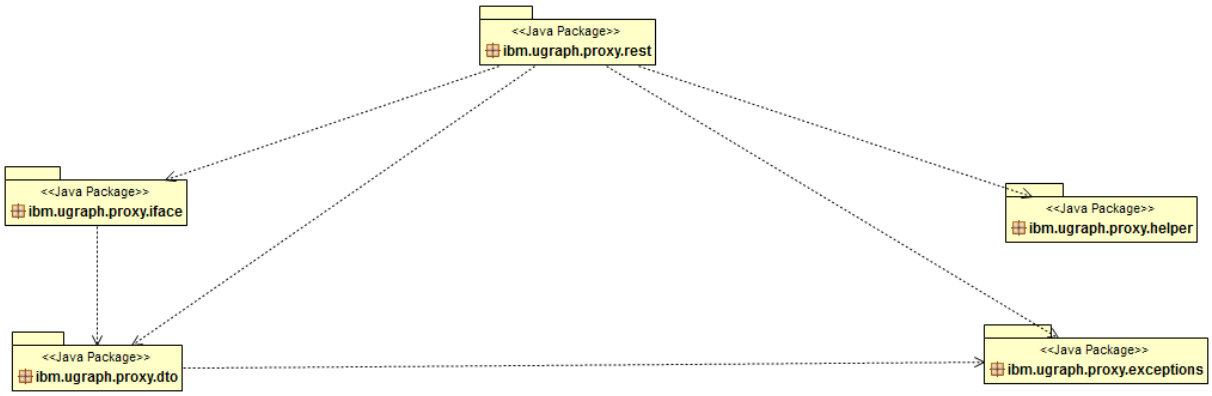


Figure 10 Package overview uGraph Proxy

4.1 Package ibm.ugraph.proxy.dto

The DTO (Data Transfer Object) package contains the Data Objects.

The Request and PluginDescriptor are needed to serialize the client request that comes in form of a JSON string from the client. For this purpose Google GSON[3] in version 2.2.4 has been used.

The TimeSeries class, on the other side, is used to serialize the data from the time series service and offers an easier way to access or manipulate data on the proxy side of the framework.

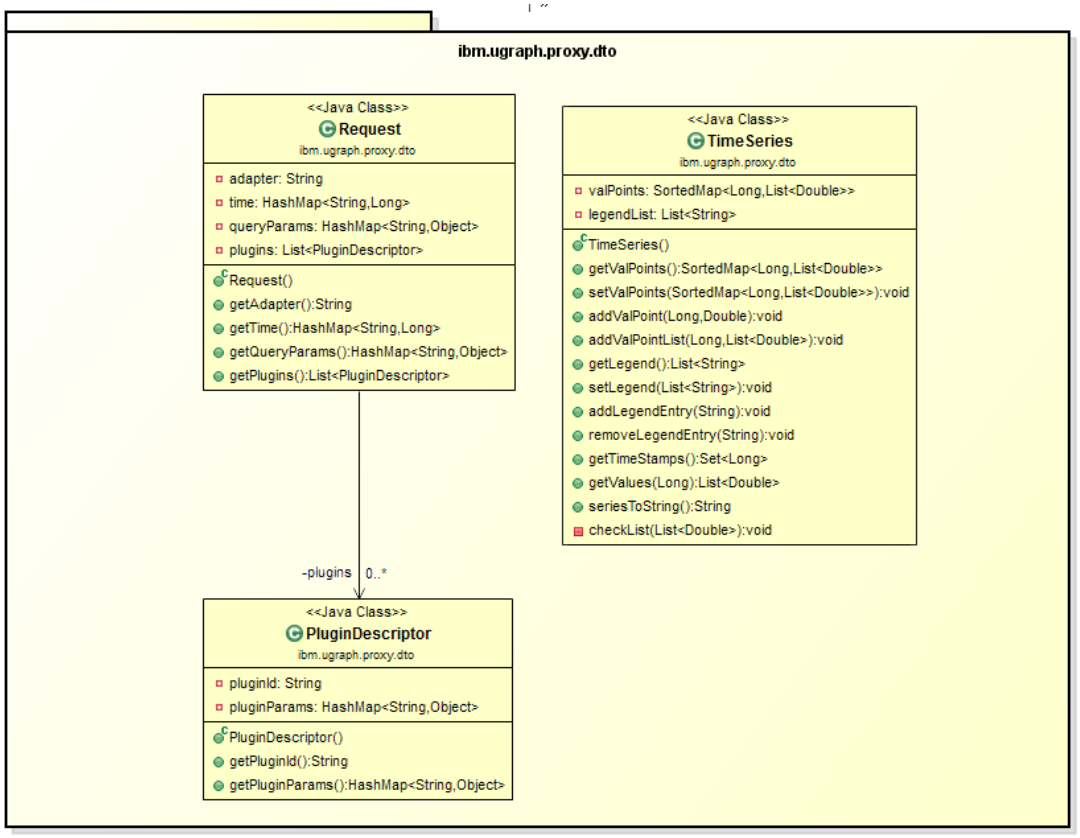


Figure 11 Overview package: ibm.ugraph.proxy.dto

4.1.1 Class: Request

As already mentioned in the intro of this section, the Request class is mainly used to make a JSON request from the client usable in Java. This is also the reason why the class offers only getter functions. This way it isn't possible for the developer to change the request parameters and start a wrong request, enhancing security inside the framework.

[3] [GooGSN]

4.1.2 Class: PluginDescriptor

The PluginDescriptor contains the description of a ProxyPlugin and has so following members:

- pluginId
- pluginParams

While the pluginId is needed for recognizing which ProxyPlugin should be loaded for handling the request the pluginParams are optional and are only used if defined for the ProxyPlugin. The need of pluginParameter depends on the implementation of the plugin.

4.1.3 Class: TimeSeries

The TimeSeries class represents the TimeSeries Object which is passed from DataAdapter to Plugin and finally to the Client. For the client, the TimeSeries Object is parsed to a JSON string. The task of the TimeSeries Object is to bring the data from the time series service to an uGraph suitable format.

For the manipulation on the server side several getter and setter functions have been created. While the function seriesToString() is responsible for creating the JSON version of the Object.

4.1.3.1 Functions

See uGraph Proxy API Documentation.

4.2 Package ibm.ugraph.proxy.iface

This package contains all the interfaces needed for the development of a DataAdapter and ProxyPlugin. The idea behind those interfaces is, to get the instance of the concrete DataAdapter or ProxyPlugin in the rest service using the Spring Framework. The concrete implementation of such a DataAdapter or ProxyPlugin is a Spring Bean. For an example of such implementation it is recommended to read the API Description of the uGraph Proxy.

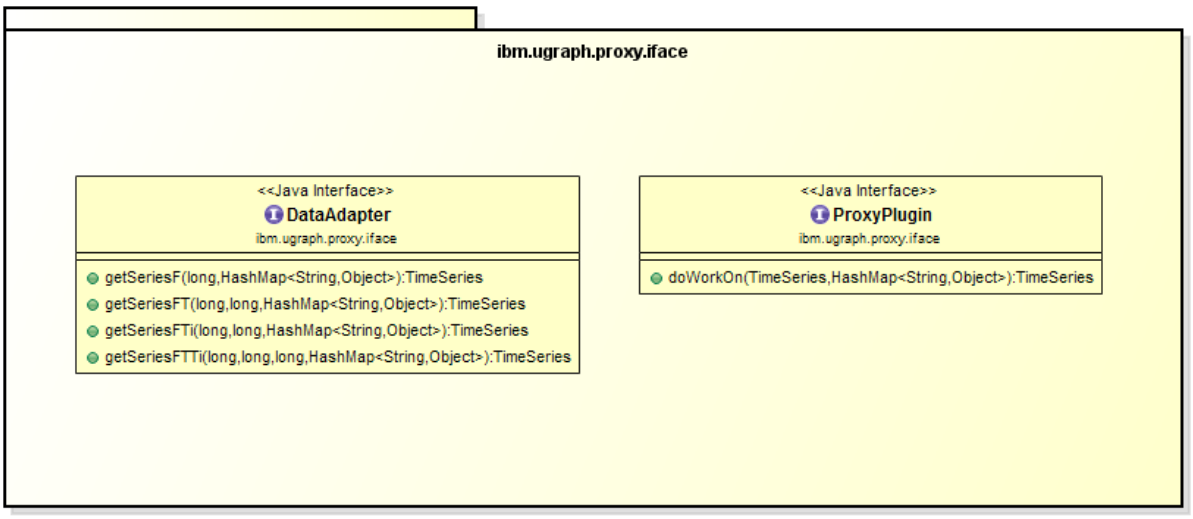


Figure 12 Overview package: ibm.ugraph.proxy.iface

4.2.1 Interface: DataAdapter

The implementation of a DataAdapter realizes the communication between Time Series service and uGraph Proxy. The implementation of this interface needs the API of the time series service to access the data. Without a DataAdapter it wouldn't be possible to access the data from the time series service.

To offer an extendable implementation of such an adapter, each function of the DataAdapter interface takes a HashMap<String, Object> with the adapter specific query parameters. This way the developer of such a DataAdapter is free to decide which parameters are optionally needed for a request to a certain time series service. Also it takes a time specification to start a request. For this purpose it has been decided to create four functions to cover every possible time declaration.

4.2.1.1 Functions

See uGraph Proxy API Documentation.

4.2.2 Interface: ProxyPlugin

The ProxyPlugin is optional, that means it is not explicitly needed for a request. It should be used for make changes on the TimeSeries Object which is generated in the DataAdapter after requesting data from a time series service. Because of this, the only function the interface owns takes two arguments:

- TimeSeries Object
- HashMap<String, Object>

The second argument is meant to offer the developer some more liberty, allowing him to take some optional parameters, like already done with the query parameters in the DataAdapter.However the ProxyPlugin is meant to be used for computationally intense operations to relieve the client.

4.2.2.1 Functions

See uGraph Proxy API Documentation.

4.3 Package ibm.ugraph.proxy.helper

The helper package contains classes that help accomplish repetitive work, making the code look cleaner.

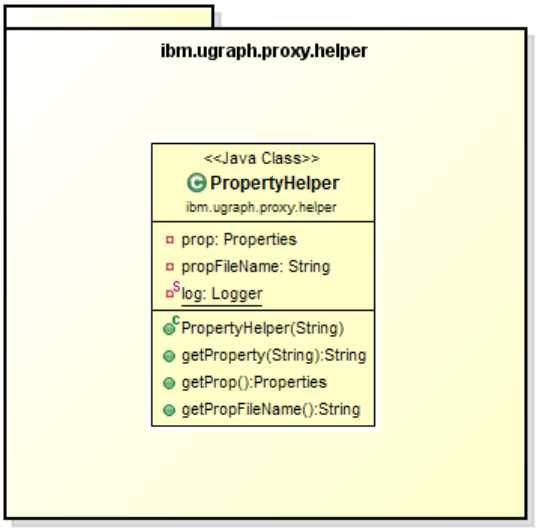


Figure 13 Overview package: ibm.ugraph.proxy.helper

4.3.1 Class: PropertyHelper

The PropertyHelper is needed for loading and managing a properties file. In the actual version of the framework, properties files are being used for internationalization only.

The error messages used on the proxy side are placed in such a properties file allowing an easy way to maintain them. With this mechanism it is also easy to switch the error messages to another language without any changes in the code.

4.3.1.1 Functions

PropertyHelper(String file)
The constructor of the PropertyHelper takes the name of the properties file as a string and loads it once into a Java Properties Object. The properties file has to be placed into the class path to be found while loading or a FileNotFoundException is being thrown and logged.

getProperty(String name)
This function takes the name of the property and returns its value.

getProp()
Return the whole Properties Object containing all the properties of the file.

getPropFileName()
Return the name of the loaded properties file.

4.4 Package ibm.ugraph.proxy.exceptions

This package contains the uGraph Proxy own exceptions. Those exceptions are mainly being thrown on failed sanity checks, informing the client and/or the developer about the error.

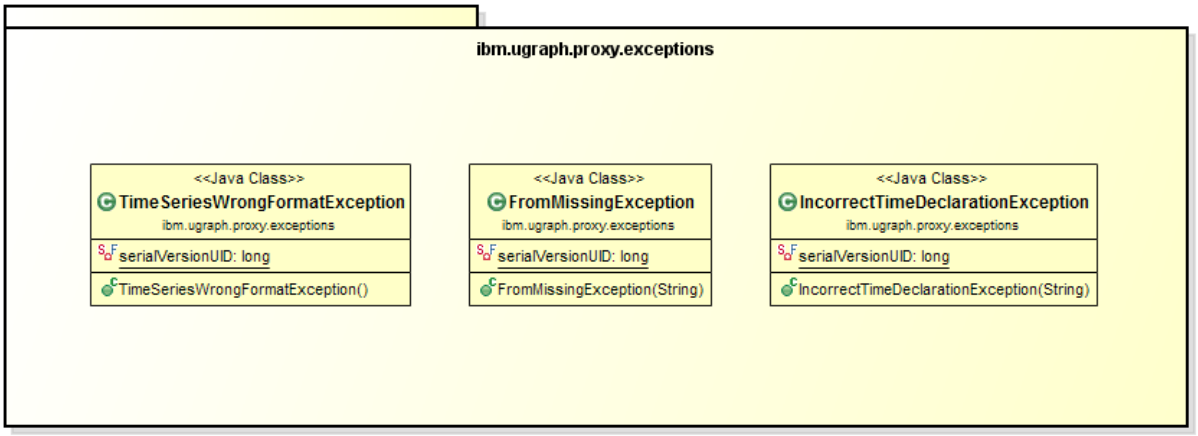


Figure 14 Overview package: ibm.ugraph.proxy.exceptions

4.4.1 Class: TimeSeriesWrongFormatException

This exception is thrown if the TimeSeries Object cannot be parsed into a JSON object that has the rightformat for the client.

4.4.1.1 Example

The following response would be an accepted by the client

```
{
	"data" : [[1386072000, 1527415.0, 1527417.0],[1386086400, 1590543.0, 15905.0]],
	"legend" : [ "x", "device1", "device2"]
}
```

The following response would not be accepted

```
{
	"data" : [[1386072000, 1527415.0, 1527417.0],[1386086400, 1590543.0, 15905.0]],
	"legend" : [ "x", "device1"]
}
```

This comes because the size of each array in the data parameter has to be as big as the size of the legend parameter. If that's not the case dygraphs isn't able to plot the data. For further information it is recommended to consult the section "TimeSeries Object" of the Proxy API documentation.

However this exception concerns only the DataAdapter developer and is not of interest for the client developer.

4.4.2 Class: FromMissingException

Taking a look at the DataAdapter interface one can see that the "from" parameter is always needed. This exception is being thrown if this parameter is missing in the client request. The exception, in this case is a fault of the requester, meaning the uGraph Client.

4.4.3 Class: IncorrectTimeDeclarationException

The IncorrectTimeDeclartionException is thrown after a sanity check on the request parameters. Before starting a request to the Time Series service the time declaration is being checked. The proxy throws this exception if the value of the "from" parameter is higher than the value of the "to" parameter and can only be provoked by the requester (uGraph Client).

4.5 Package ibm.ugraph.proxy.rest

The rest package contains the controllers needed for the handling of the client requests. While the RequestHandler contains the REST functionality, the PluginController and the AdapterController are used to do specific operations, meaning that those last two classes contain the business logic while the RequestHandler is only responsible for the communication with the client.

However for a better understanding of this section it is recommended reading the REST API section of the Proxy API description first.

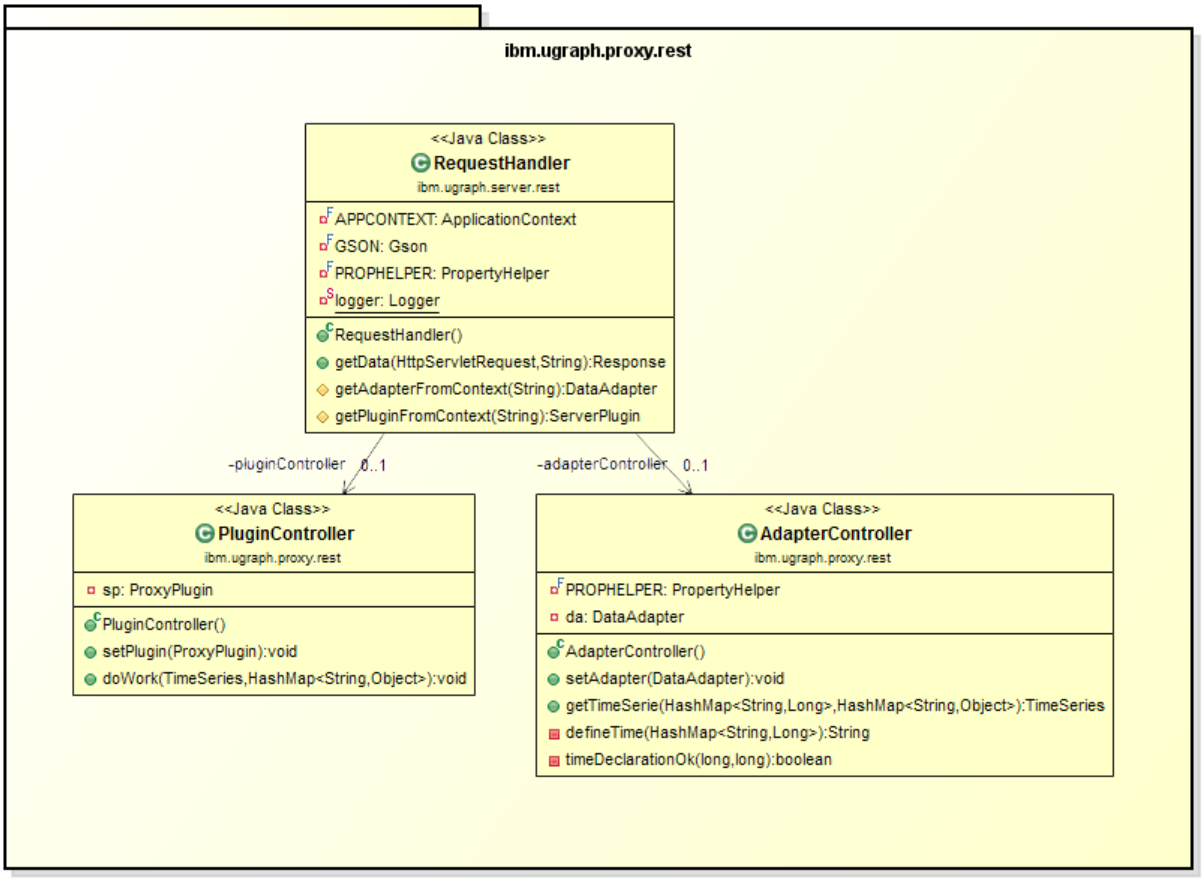


Figure 15 Overview package: ibm.ugraph.proxy.rest

4.5.1 Class: RequestHandler

The RequestHandler is meant to coordinate the client requests and answer them. The most important function of the RequestHandler is the error handling. Every exception thrown on the proxy side of the framework is passed thru till the RequestHandler which then decides what to do. In normal case it logs the error message to a file, using Log4J and informs the client about the exception.

4.5.1.1 Responsibilities

- Initializing the Spring ApplicationContext responsible for the loading of Spring Beans (DataAdapter and ProxyPlugins)
- Initializing an instance of GSON for serialize the JSON requests sended by the client.
- Initializing an instance of the PropertyHelper to get the error messages from the properties file.
- Calling the right DataAdapter instance and setting it for the AdapterController.
- Calling the right ProxyPlugin instance and setting it for the PluginController.
- Handle Exceptions
    - Log error messages.
    - Forward them to the client.

4.5.1.1.1 Spring Beans and Application Context

A Spring Bean is an object that is instantiated and managed by the Spring Framework. All object instances in the Spring Framework are called Beans. The Spring Framework offers containers which are responsible for the management of such a Bean. There are two types of Container, the BeanFactory and the ApplicationContext.

> *"The BeanFactory is a Simple container supporting basic dependency injection, whereas ApplicationContext is an extension of BeanFactory, which has few additional bells and whistles.*
> *In my personal opinion, I would vote for using ApplicationContext over BeanFactory…" [MadSPR]*

uGraph Proxy uses the ApplicationContext. There is more than one implementation of such an ApplicationContext, for the uGraph Proxy the ClassPathXMLApplicationContext is used. This container reads the XML configuration file of the Spring Framework directly from the classpath. This implementation allows having the bean file in a JAR file.

4.5.1.2 Functions

getData(HttpServletRequest requestId, String body)
The getData function is the function that can be reached by calling the REST service URI. For more information check the uGraph Proxy API Description.

getAdapterFromContext(String adapterName)
Returns the instance of the DataAdapter with help of the Spring AppContext. This function is written mainly for testing purposes.

getPluginFromContext(String pluginName)
Returns the instance of the ProxyPlugin with help of the Spring AppContext. This function is also written mainly for testing purposes.

4.5.2 Class: AdapterController

The AdapterController is responsible for the decision of DataAdapter function that has to be called. To do so the time declaration has to be defined. While defining the time declaration the time parameter passed by the client will be checked on errors.

The AdapterController has been separated from the RequestHandler to allow dependency injection, injecting any DataAdapter using the setAdapter() function.

4.5.2.1 Responsibilities

- Define time declaration
  - Checking if the request is valid by validating the parameters or throw exception and inform the client
- Call the DataAdapter function
- Return the TimeSeries Object to the RequestHandler

4.5.2.2 Used Patterns

The adaptation to the right DataAdapter has been realized using the Adapter-Pattern[4], allowing testing by using mock adapters.

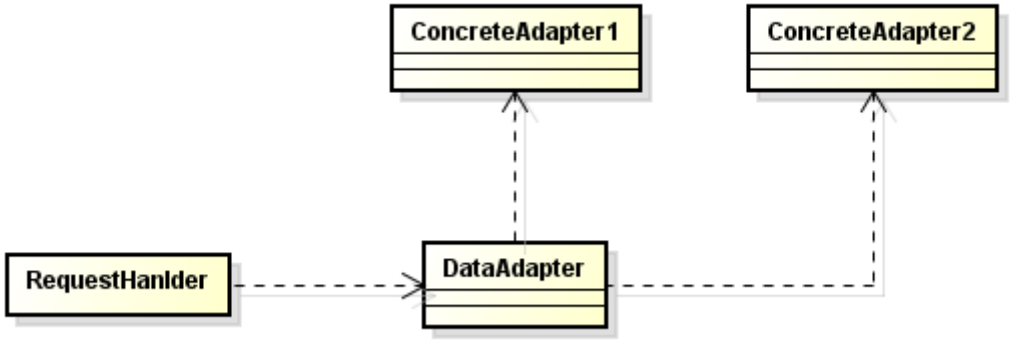


Figure 16 Adapter Pattern UML

4.5.2.3 Functions

setAdapter(DataAdapter da)
The function sets the DataAdapter which should be called for the Request. This function is only called once per request by the RequestHandler for setting the requested DataAdapter.

defineTime(HashMap<String, Long> time)
The function defines which time parameters have been given in the request by checking the content of the HashMap and returning a string as identifier. Possible content of the HashMap is:

- "from" : <utc_timestamp>;
- "from" : <utc_timestamp>; "to" : <utc_timestamp>;
- "from" : <utc_timestamp>; "ticksize" : <long>;
- "from" : <utc_timestamp>; "to" : <utc_timestamp>; "ticksize" : <long>

This function throws a FromMissingException if the "from" parameter is not given.

timeDeclarationOK(long from, long to)
Is used only if the time definition is "from, to" or "from, to, ticksize". This function returns a boolean (true) if the "from" value is smaller than the "to" value and throws IncorrectTimeDeclarationException if not.

getTimeSeries(HashMap<String, Long> time, HashMap<String, Object> queryParams)
The function calls the corresponding function of the DataAdapter. The function defineTime() helps defining which function of the DataAdapter should be called.

[4] [LarADA]

4.5.3 Class: PluginController

The PluginController has the same principle of the AdapterController. It is used to coordinate ProxyPlugin operations. The principle of dependency injection has also been used in the same way using the setPlugin() function to inject the ProxyPlugin instance.

4.5.3.1 Responsibilities

- Call the doWorkOn() function of the ProxyPlugin.

4.5.3.2 Functions

setPlugin(ProxyPlugin sp)
Sets the ProxyPlugin used for manipulate the TimeSeries Object.

doWork(TimeSeries ts, HashMap<String, Object> pluginParams)
Runs the doWorkOn() function of the ProxyPlugin.

4.6 Architectural decisions

4.6.1 Use of the Spring Framework

4.6.1.1 The Problem

The framework should be extendable, meaning that additional Time Series services can be added at any time.

4.6.1.2 Constraints

- Java as programming language
- Works with JAX-RS/Jersey

4.6.1.3 Treats

- Extendable security
    - Authentication

4.6.1.4 Tested alternatives

Before working with the Spring Framework a prototype was written. In the first prototype of the uGraph Proxy, JNDI has been used for loading dynamically JARs into the proxy by accessing it directly on the file sytem and load them at runtime into the class path.

- JNDI
    - Pro: JAR files can be loaded without restarting the server
    - Contra: Direct access on file system allows to load any JAR into the framework (security issue)

4.6.1.5 Decision

Although JNDI offers infinite liberty for the loading of JARs and also hot-plug ability it has been decided to switch to the Spring Framework. The security issue can so be removed with low costs furthermore the Spring Framework works perfectly together with JAX-RS which is used for the REST service.

4.6.2 POST instead of GET for the REST call

4.6.2.1 The Problem

Large JSON string as request but parameters can only be transmitted via URL which is limited to > 2000 chars.

4.6.2.2 Constraints

- REST Service.
- Length of the JSON cannot be shortened.

4.6.2.3 Tested alternatives

- Passing the arguments as URL parameters
  - Pro: Keep the right http method for getter calls
  - Contra: Unexpected errors can occur

4.6.2.4 Decision

To solve the problem of the unexpected errors that could occur passing the arguments via URL, it has been decided to send the request parameters in the body of a POST request, although the right method according to w3.org would be GET.

> *"The GET method means retrieve whatever information (in the form of an entity) is identified by the Request-URI. If the Request-URI refers to a data-producing process, it is the produced data which shall be returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process." [w3oGET]*

4.6.3 GSON for serializing JSON strings

4.6.3.1 The Problem

Serializing the JSON request (string) into a Java object.

4.6.3.2 Constraints

- The used serialization library should be easy to replace with another.

4.6.3.3 Tested alternatives

- Jackson[5]

4.6.3.4 Decision

GSON offers a simple way to serialize JSON string into Java classes. The GSON library also contains additional data structures which help a lot mapping single parts of the JSON to a Java object. The partner IBM also tested some ways to serialize such JSON object using different libraries and found that GSON has the better performance than Jackson. For that reason it has been decided to use GSON for that case.

[5] [CodJAC]

5 Testing

5.1 uGraph Client

5.1.1 Used testing framework

To test the client's JavaScript code the QUnit testing framework was used. QUnit is a light weight, easy-to-use JavaScript unit testing framework. The QUnit framework was chosen because it is easy to use and provides a nice user interface.

5.1.2 Unit Tests

Unit tests were written for all the modules except the uGraph.ui module (see 5.1.2.2 uGraph.ui).

5.1.2.1 uGraph.bl testlog

The uGraph.bl module relies on data from the uGraph.dl module. To be able to test the bl module a fake uGraph.dl had to be made, that supplies the same data every time the test is run. The fake dl has the same functions like the real dl but with a different implementation. In the fake dl all the data needed to be static to make sure the test always runs on the same data. The faked dl was then passed to the uGraph.bl via its constructor.

5.1.2.2 uGraph.ui

For the uGraph.ui module no unit tests were written because it has many dependencies and it would have exceeded the project scope to fake all the other modules and find a way to test if dygraphs plotted the correct graph.

5.2 uGraph Proxy

The uGraph Proxy was tested using JUnit like usual for Java Applications.

The uGraph Proxy Testing has been split into two packages. The package ibm.ugraph.proxy.test.unit contains all the test cases, while the package ibm.ugraph.proxy.test.mocks offers the mocking objects for realizing the tests.

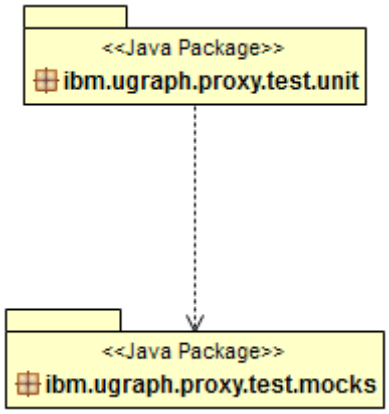


Figure 17 Overview testing packages

5.2.1 Package ibm.ugraph.proxy.test.mocks

The package implements mocks for the Testing of the DataAdapter and the ProxyPlugin. Beside those two mocks, a MockRequestHandler has been created. The MockRequestHandler was needed for testing the class RequestHandler without loading explicit Spring Beans declared in the application context of the Spring Framework.

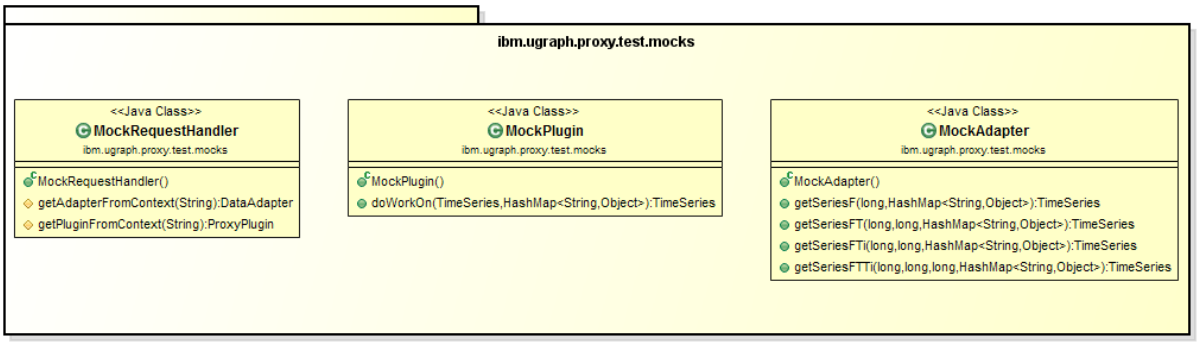


Figure 18 Overview package ibm.ugraph.proxy.test.mocks

5.2.1.1 Class MockRequestHandler

The function of the RequestHandler getAdapterFromContext() and getPluginFromContext() have been changed in the MockRequestHandler, allowing to load the MockAdapter or MockPlugin for testing purposes.

5.2.2 Package ibm.ugraph.proxy.test.unit

The tests for the Controller and RequestHandler have been split into separate test cases, while the DataTransferObjects have been tested in a general test case.

The test cases of the DTOs contain mainly function to test the TimeSeries Object. Other DTO are not being tested because of the complexity, since they contain only getter functions.

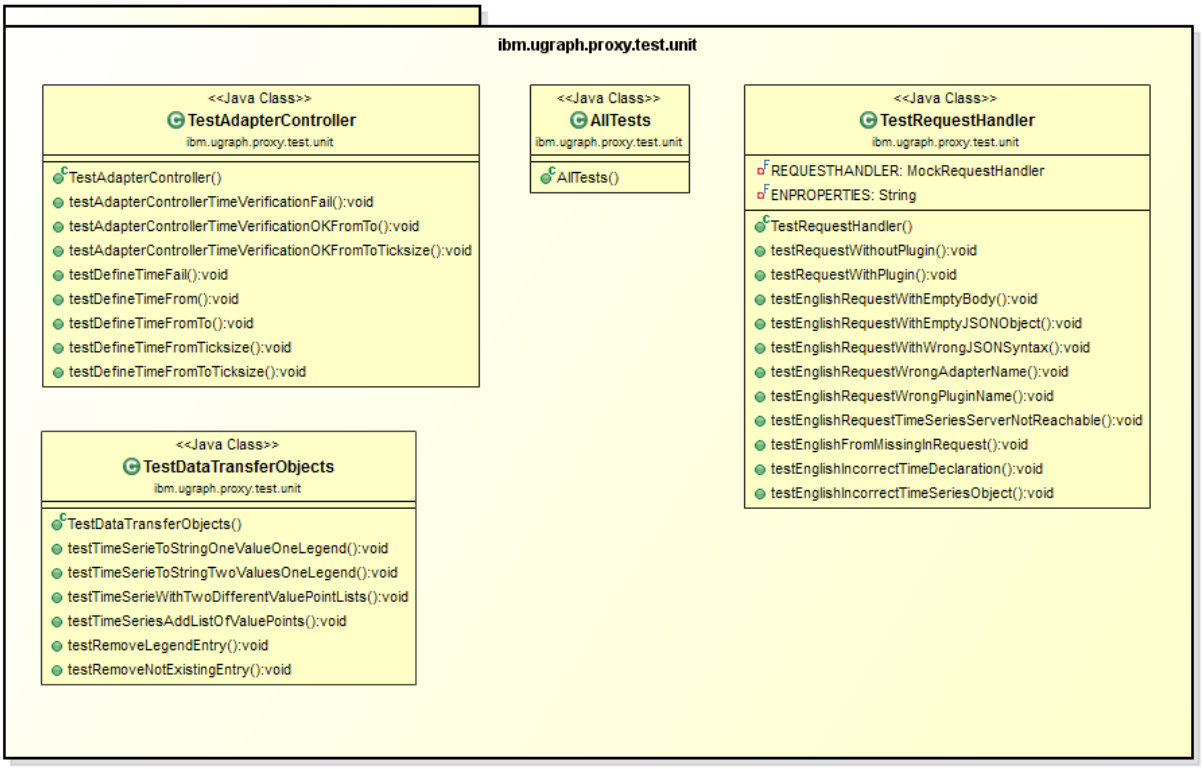


Figure 19 Overview package ibm.ugraph.proxy.test.unit

5.2.3 Test coverage

With the test cases of the package ibm.ugraph.proxy.unit, a test coverage of 95.6% has been achieved.

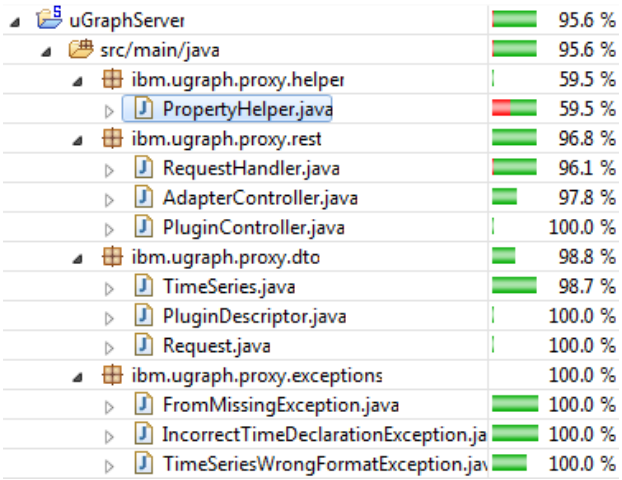


Figure 20 Test coverage uGraph Proxy

6 Glossary

Time Series Service	A Database or web service which offers an interface for requesting Time Series Data
Plotting Library	Library for the visualization of Time Series Data in a Graph
TimeSeries Object	The TimeSeries Object is a Data Transfer Object. It is created by the DataAdapter and is passed to the Controller and Plugins.
Dygraphs	The open source plotting library used for this project. http://dygraphs.com/
Google GSON	Java Library for serializing from/to JSON.
Value Point	A single point in the graph matched to a date.
Log4J	Library used for logging purposes.
QUnit	The JavaScritp testing framework used to test the client code. http://qunitjs.com/
DTO	Data Transfer Object
JAR	Java Archive

7 Bibliography

Abbreviation	Source
[w3oGET]	w3.org. (2014, June 9). Retrieved from http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html
[OlaMW]	Zimmermann, O. (2014, June 9). Middleware and Layering.
[AddRMP]	Osmani, A. (2014, June 10). *Learning JavaScript Design Patterns*. Retrieved from http://addyosmani.com/resources/essentialjsdesignpatterns/book/#revealingmodulepatternjavascript
[BusRLP]	Layers Pattern: Relaxed Layered System. (2014). In F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, & M. Stal, *Pattern - Oriented Software Architecture: A System Of Patterns* (pp. 31 - 51). Rapperswil: John Wiley & Sons.
[GooGSN]	Inc., G. (2014, June 06). *google-gson*. Retrieved from https://code.google.com/p/google-gson/
[CodJAC]	Codehaus. (2014, June 10). *Jackson*. Retrieved from http://jackson.codehaus.org/
[LarADA]	Larman, C. (2005). *UML2 und Patterns angewendet.* mitp.
[MadSPR]	Konda, M. (2011). Container. In M. Konda, *Just Spring* (p. 29). O'REILLY.

Client API

Giuseppe Aquino & Simon Brouwer

1 Contents

1. Introduction

1.1 Purpose

This document should help the developer understand the client side API of the uGaph Framework. It also gives a description of the options that can be set for the uGraph Framework.

2 Options Reference

2.1 Introduction

In this chapter the options which can be set for the framework are explained. Due to the fact that uGraph wraps the dygraphs library all the options that can be set in the dygraphs library are also available in the uGraph framework. In this chapter only the functions that are newly added by uGraph are explained, the dygraphs options can be found on their website.

2.2 autoUpdate

Time interval in which new data is dynamically loaded from the proxy and shown in the diagram.

Type: integer

Unit: seconds

Default: 0

2.2.1 Example

New data will be loaded every five seconds.

```
var plottingOptions = {
      autoUpdate : 5
};
```

2.3 movingWindow

To be used in cooperation with the autoUpdate option. If set to true the "from" time is also updated when loading new data, this allows for loading new data without overcrowding the diagram. Has no effect if the autoUpdate Option is not active.

Type: Boolean

Default: false

2.4 Example

New data will be loaded every five seconds. Moving Window is activated.

```
var plottingOptions = {
      autoUpdate : 5,
      movingWindow : true
};
```

2.5 updateOnPan

If set, when panning the diagram out of the range of data, new data is loaded form the proxy into the diagram when panned further than the range specified here.

Type: integer

Unit: seconds

Default: 0

2.5.1 Example

If panning more than one day, new data is loaded.

```
var plottingOptions = {
      updateOnPann : 86400
};
```

2.6 preloadDataLeft & preloadDataRight

Specifies how much data is loaded earlier(left) or past(right) the time in the request. Only the data set in the request time is displayed in the diagram, the data loaded left and/or right is loaded into the diagram but is only shown after panning to the left or right. Can be used in conjunction with the updateOnPan option to make sure data is always shown while panning.

Type: integer

Unit: seconds

Default: 0

2.6.1 Example

Data for one day before the start date and one day after the end date will be loaded.

```
var plottingOptions = {
      preloadDataLeft : 86400,
      preloadDataRight: 86400
};
```

2.7 updateOnZoom

If zooming and the shown time range is smaller than the updateOnZoom value, higher resolution data for that range is dynamically loaded from the server and shown in the diagram.

Type: integer

Unit: seconds

Default: 0

2.7.1 Example

If zoomed to only show one day, new data is loaded.

```
var plottingOptions = {
      updateOnZoom : 86400
};
```

2.8 updateOnZoomMultiplier

To be used in conjunction with the updateOnZoom option. When loading new data after zooming, the updateOnZoom value is multiplied by this factor. This makes sure when zooming more and more the updateOnZoom value also gets "zoomed" and prevents from firing updateOnZoom events too often.

Type: integer

Default: 0.5

2.8.1 Example

UpdateOnZoom value will be multiplied by 0.2 after the updateOnZoom event fired. Now new data is loaded when zooming in to show about five hours.

```
var plottingOptions = {
      updateOnZoom : 86400,
      updateOnZoomMultiplier : 0.2
};
```

2.9 ZoomTickMultiplier

To be used in conjunction with the updateOnZoom option. This allows for specifying how much more detailed the newly loaded data should be. The ticksize value in the time parameter is multiplied with this value when new data is loaded after zooming.

Type: integer

Default: 0.5

2.9.1 Example

Ticksize, in the time object, will be multiplied by 0.2 when the updateOnZoom event fired. Distance between ticks will be four times smaller.

```
var plottingOptions = {
      updateOnZoom : 86400,
      updateOnZoomMultiplier : 0.2,
      zoomTickMultiplier : 0.25
};
```

2.10 updateOnPanMultiplier

To be used in conjunction with the updateOnZoom option. Makes sure that updateOnPan value gets adjusted to the zoom level. Prevents the user from having to pan out of the range of data being shown when zoomed.

Type: integer

Default: 0.5

2.10.1 Example

UpdateOnPann will be multiplied by 0.2 after the updateOnZoom event fired. Now only five hours have to be panned to load new data.

```
var plottingOptions = {
      updateOnPann : 86400,
      updateOnZoom : 86400,
      updateOnZoomMultiplier : 0.2,
      zoomTickMultiplier : 0.25,
      updateOnPanMultiplier : 0.2
};
```

3 API Reference

3.1 uGraph.Client

Constructor of the uGraph library.

Parameter	type	description
graphElement	DOM Element	Div to draw the diagram in
legendElement	DOM Element	Optional. Div to draw the legend in. If not specified the legend will be drawn in the diagrams canvas.
url	String	Optional. URL to where the uGraph server is located. Default : http://localhost:8080/uGraphServer/rest/data/get-data/

3.1.1 Example

```
var ugraph = new uGraph.Client(document.getElementById("grap_div"),
      document.getElementById("legend_div"),
      http://localhost:8080/uGraphServer/rest/data/get-data/");
```

3.2 getGraph

Initial function to draw a diagram.

Parameter	type	description
adapter	String	The identifier of the adapter to be called
time	Object	Time object with time range to get the data in
queryParameter	Object	Object with parameters for the adapter to use
serverPlugins	Object	Array of objects with identifier for server-side plugins and the corresponding parameters
plottingOptions	Object	Object with options for specifying the behavior and look of the diagram

3.2.1 time

The time object consists of all the information needed to specify a range of time in which the time series data has to be loaded. It is a plain javascript object containing three members. An example can be found in the code snippet below.

member	type	unit	description
from	UTC timestamp	seconds	UTC timestamp that specifies the start time of the time range.
to	UTC timestamp	seconds	Optional. UTC timestamp that specifies the end time of the time range. If not specified the adapter should return data to the current date and time.

ticksize	Integer	seconds	Optional. Size of one tick in seconds. Can be used to manipulate how many data points are loaded.

3.2.2 queryParameter

In the queryParameter object any parameters can be defined, these parameters are then sent to the proxy and passed on to the adapter where, for example, they can be used to build the query to the time series data server. An example can be found in the code snippet below.

3.2.3 serverPlugins

In the serverPlugins array, for each plugin to be executed, all the information that the proxy needs to run a server side plugin is stored. An example can be found in the code snippet below.

member	type	description
pluginId	String	Identifier for the plugin to be run
pluginParams	Object	Object with custom parameter that may be needed by the plugin. Can be an empty object.

3.2.4 plottingOptions

The plottingOptions object contains all the options that specify the look and behavior of the diagram. In this object all the options that dygraphs supports standardly as well as the options added by uGraph can be set. Further information can be found in the Options Reference Chapter.

3.2.5 Example

```
var ugraph = new uGraph.Client(document.getElementById("grap_div"), document
            .getElementById("legend_div"));

var time = {
      from : 1400199900,
      to : 1401190513,
      ticksize : 1200
};
var adapter = "adapter1";

var queryParams = {
      param1 : "param1",
      param2 : "param2"
};

var serverPlugins = [ {
      pluginId : "average",
      pluginParams : {
            description : "Server Average"
      }
} ];

var plottingOptions = {
      updateOnPann : 86401,
      preloadDataLeft: 86400
};

ugraph.getGraph(adapter, time, queryParams, serverPlugins, plottingOptions);
```

3.3 setCanvas

Setter for the div elements to draw the diagram and the legend in

Parameter	type	description
graphElement	DOM Element	Div to draw the diagram in
legendElement	DOM Element	Optional. Div to draw the legend in. If not specified the legend will be drawn in the diagrams canvas.

3.3.1 Example

```
ugraph.setCanvas(document.getElementById("new_graph_div"),
document.getElementById("new_legend_div"));
```

3.4 setAdapter

Setter for the adapter identifier.

Parameter	type	description
adapterId	String	The identifier of the adapter to be called

3.4.1 Example

```
ugraph.setAdapter("newAdapter");
```

3.5 setTime

Setter for the time range in which data should be fetched.

Parameter	type	description
Time	Object	Time object with time range to get the data in. See Chapter 3.2.1 time for a detailed description of the time object.

3.5.1 Example

```
var time = {
      from : 1400199900,
      to : 1401190513,
      ticksize : 1800
};

ugraph.setTime(time);
```

3.6 setQueryParameter

Setter for the query parameters that are sent to the server and are passed to the called adapter.

Parameter	type	description
queryParameter	Object	Object with parameters for the adapter to use

3.6.1 Example

```
var queryParams = {
      param1 : "param1",
      param2 : "param2"
};

ugraph.setQueryParameter(queryParams);
```

3.7 setServerPlugins

Setter for the list of Plugins to be executed on the server and their parameters.

Parameter	type	description
serverPlugins	Object	Object with identifiers for server-side plugins and the corresponding parameters

3.7.1 Example

```
var serverPlugins = [ {
	pluginId : "average",
	pluginParams : {
		description : "Server Average"
	}
}, {
	pluginId : "complexCalculation",
	pluginParams : {
		param1 : "param1",
		param2 : "param2"
	}
} ];

ugraph.setServerPlugins(serverPlugins);
```

3.8 addServerPlugin

Adds a server plugin object to the list of plugins to be executed on the server.

Parameter	type	description
Plugin	Object	Object with identifier for server-side plugin and the corresponding parameters

3.8.1 Example

```
var serverPlugin = [ {
	pluginId : "complexCalculation2",
	pluginParams : {
		param1 : "param1",
		param2 : "param2"
	}
} ];
ugraph.addServerPlugin(serverPlugin);
```

3.9 removeServerPlugin

Removes a server plugin object from the list of plugins to be executed on the server.

Parameter	type	description
PluginId	String	Identifiert for the plugin to be removed

3.9.1 Example

```
ugraph.removeServerPlugin("complexCalculation2");
```

3.10 registerClientPlugin

Register a plugin function with the uGraph object. The registered plugins will not yet be executed when querying for data. They have first to be added to the list of client plugins to be executed, see XY addClientPlugin().

Parameter	type	description
id	String	Identifier for the plugin
func	Function	The implementation of the plugin.

3.10.1 Example

In this example a client side plugin, which calculates an average value over the different time series, is registered with ugraph.

```
clientPlugin = function(data) {
	data.addLegendElement("Average");

	var tstamps = data.getTimestamps();
	$.each(tstamps, function(key, value) {
		var sum = 0;
		var values = data.getValues(value);
		for ( var i = 0; i < values.length; i++) {
			if (values[i] != null) {
				sum = sum + values[i];
			}
		}
		data.addValue(value, sum / values.length);
	});
	return data;
};
ugraph.registerClientPlugin("Average", clientPlugin);
```

3.11 deregisterClientPlugin

Deregisters a plugin from the uGraph object.

Parameter	type	description
id	String	Identifier for the plugin to be removed

3.11.1 Example

```
ugraph.deregisterClientPlugin("Average");
```

3.12 addClientPlugin

Add a client plugin from the list of registered plugins to the list of plugins to be executed after having fetched the data from the proxy.

Parameter	type	description
id	String	Identifier for the plugin to be added to the list.

3.12.1 Example

```
ugraph.addClientPlugin("Average");
```

3.13 removeClientPlugin

Removes a client plugin from the list of plugins to be executed. The plugins will stay registered.

Parameter	type	description
id	String	Identifier for the plugin to be removed from the list.

3.13.1 Example

```
ugraph.removeClientPlugin("Average");
```

3.14 setPlottingOptions

Add new PlottingOptions or override old ones. The options that have been set before will be conserved. Options that were already set will be overwritten.

Parameter	type	description
plottingOptions	Object	Object with options for specifying the behavior and look of the diagram

3.14.1 Example

```
var plottingOptions = {
      updateOnPann : 0,
      preloadDataLeft : 0,
      autoUpdate : 5
};

ugraph.setPlottingoptions(plottingoptions);
```

3.15 updateGraph

Gets new data from the proxy and updates the diagram. Is useful when proxy related options, like "queryParameters" or "time", have been changed and new data with these updated parameters has to be loaded.

3.15.1 Example

```
ugraph.updateGraph();
```

3.16 redraw

Redraws the graph. Is useful when client related options, like "plottingOptions", have been changed and the diagram has to be updated accordingly.

3.16.1 Example

```
ugraph.redraw();
```

3.16.2 Complete Example

This Example shows a complete Web Page with everything necessary to draw a graph with the uGraph Framework.

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="ISO-8859-1">
		<title>Demo</title>
		<link href="bootstrap/css/bootstrap.min.css" rel="stylesheet"
		type="text/css">
		<style>
			body {
				font-family: Verdana;
				font-size: 12px;
			}

			h1 {
				margin-left: 65px;
			}

			#content {
				margin: 0px auto;
				width: 850px;
			}

			.col-md-8 {
				height: 250px;
				margin-bottom: 20px;
			}
		</style>
	</head>
	<body>
		<div id="content">
			<h1>uGraph</h1>
			<div class="row">
				<div class="col-md-8" id="g1">
					.col-md-8
				</div>
				<div class="col-md-4" id="l1">
					.col-md-4
				</div>
			</div>
		</div>
```

```
<script type="text/javascript" src="../libs/jquery-1.11.0.min.js"></script>
		<script type="text/javascript"
src="bootstrap/js/bootstrap.min.js"></script>
		<script type="text/javascript" src="../libs/dygraph-
combined.js"></script>
		<script type="text/javascript"
		src="../libs/util/ibm.ugraph.util.utils.js"></script>
		<script type="text/javascript"
		src="../libs/bl/ibm.ugraph.bl.timeSeries.js"></script>
		<script type="text/javascript"
		src="../libs/dl/ibm.ugraph.dl.dataContext.js"></script>
		<script type="text/javascript"
src="../libs/ui/ibm.ugraph.ui.graph.js"></script>
		<script type="text/javascript"
src="../libs/ibm.ugraph.main.js"></script>

		<script type="text/javascript">
			(function($, window, document) {
				var g = new uGraph.Client(document.getElementById("g1"),
document.getElementById("l1"));

				var time = {
					from : 1400199900,
					to : 1401190513,
					ticksize : 1200
				};
				var adapter = "adapter1";
				var queryParams = {
					param1 : "param1",
					param2 : "param2"
				};
				var serverPlugins = [{
					pluginId : "averate",
					pluginParams : {
						param1 : "param1"
					}
				}];
				var plottingOptions = {};
				g1.getGraph(adapter, time, queryParams1, serverPlugins,
plottingOptions);
			})();
		</script>

	</body>
</html>
```

4 Time Series Data Description

The time series data that the uGraph Client gets from the proxy is a format that the plotting library dygraphs can use and is also stored on the client in that format so that it does not have to be parsed again for the plotting library to be able to use it. The data format is described in detail below.

If the user of the uGraph Framework writes a plugin, the format that dygraphs uses is not optimal for manipulating the data, in order to make working on the time series data easier an abstraction of the data is created and the plugin works on that abstraction of the time series data. This data format is also described in detail below.

4.1 Dygraphs data format

In order for the dygraphs library to be able to render the diagram the data has to be formatted in a specific way. The time series data has to be stored in a JavaScript object, containing a "data" member with the time series data and a "legend" member with the corresponding legend entries.

The data member is an array of arrays. Each array contains a timestamp at the first position and then a value for each graph.

The legend member is an array of strings. At the first position it always contains the "x" element to specify that the first value of the data array is to be interpreted as the timestamp. Every next string is the legend entry for the value in the corresponding array-position in the data array.

4.1.1 Example

This example would show two graphs with three values each in the diagram. One graph would be called "graph1" and the other one "graph2". The "graph1" contains the values 1, 2 and 3. The "graph2" contains the values 5, 3, 1.

```
{
	data : [[1400198400, 1, 5],[1400200200, 2, 3],[1400202000, 3, 1]],
	legend : ["x", "graph1", "graph2"]
}
```

4.2 uGraph.bl.abstractTimeSeriesData

The abstracTimeSeriesData makes it easier to manipulate the time series data on the client. It does not make a copy of the time series data but stores the timestamps and the corresponding index in the time series array to access the values. This allows the user to directly access the values of a timestamp without iterating over the whole time series array. The functions that the module abstractTimeSeriesData offers are described in detail below.

4.2.1 getTimestamps

Returns an array with all the timestamps in the time series object. Useful when getting or adding values form/to the time series object.

4.2.1.1 Example

```
var tstamps = data.getTimestamps();
```

4.2.2 getLegend

Returns an array with the legend entries.

4.2.2.1 Example

```
var legend = data.getLegend();
```

4.2.3 addLegendElement

Adds a legend entry to the existing list of legend entries.

Parameter	type	description
legendElement	String	String with the legend entry for a newly added graph

4.2.3.1 Example

```
data.addLegendElement("newGraph");
```

4.2.4 getValues

Returns an array with the graph values for a given timestamp.

Parameter	type	description
timestamp	integer	Timestamp at which to get the values of the graphs.

4.2.4.1 Example

```
data.getValues(1400599900);
```

4.2.5 addValue

Adds a new value to the existing list of values for a given timestamp.

Parameter	type	description
timestamp	integer	Timestamp on which to add the new value
Value	Integer	Value to be added

4.2.5.1 Example

```
data.addValue(1400599900, 5);
```

4.2.6 addSerie

Adds a whole time series to the existing time series. Corresponds to adding a new graph to the diagram. The addes series has to be formatted exactly like the dygraphs data format and the timestamps have to exactly match the timestamps of the existing time series otherwise the data is not added, because dygraphs would not be able to plot the data.

Parameter	type	description
serie	object	Object with the series data. Has to be formatted exactly like the dygraphs data format documented above. Can only contain one new value per timestamp.

4.2.6.1 Example

```
data.addSerie({data : [[1400599900, 3][1400600000, 5]][1400699900, 9],
 legend : ["x","newGraph"]});
```

5 Glossary

Time Series Service	A Database or web service which offers an interface for requesting Time Series Data
Plotting Library	Library for the visualization of Time Series Data in a Graph
TimeSeries Object	The TimeSeries Object is a Data Transfer Object. It is created by the DataAdapter and is passed to the Controller and Plugins.
Dygraphs	The open source plotting library used for this project. http://dygraphs.com/

Proxy API

user

1 Contents

1. Introduction

1.1 Purpose

This document should help the developer understand the classes and interfaces needed to extend the uGraph Proxy by adding DataAdapters and ProxyPlugins. Also it gives a description of the REST API.

1.2 References

- Solution document

1.3 General information for the implementation of DataAdapters/ProxyPlugins

The interfaces for the implementation of a DataAdapter or ProxyPlugin are contained in the Eclipse Project of the proxy. To access those interfaces, the uGraph Proxy project has to be added to the build path.

2 REST API

Following the list of the resources of the uGraph Proxy.

Resource	Description
POST rest/data/get-data	Returns the requested data, including a legend, for the requested time range.

2.1 POST rest/data/get-data

2.1.1 Resource information

Authentication	Not required
Response Formats	JSON
HTTP-Methods	POST
Response Object	TimeSeries

2.1.2 Parameters

2.1.2.1 Request

The parameters for this resource have to be submitted in the body of the request.

adapter	Identifier of the adapter as string. Adapter has to already exist on server.
time	Object containing the dime declaration for the request. Possible Time Parameters: • from • from, to • from, ticksize • from, to, ticksize
queryParams	Object containing the query parameters for the adapter. The parameter name depend on the adapter implementation and may vary.
plugins	Array of strings, whereby the string is the identifier for the plugins.

2.1.2.2 Response

data	Array of arrays. The Arrays containing the values are structured as follow: • First element index 0 corresponds to a utc timestamp • All other values are actual values Each list has the same size as the legend array.
legend	The legend is an array of string. The first entry (index 0) is needed for the plotting library dygraphs to display the time given by the timestamp on the graph. All the following entries are legend entries that can later be displayed on the graph.

2.1.2.3 Example request

In Example 1 a simple request is shown. The Time Series Data is requested form the time set in the parameter "from" to the current date, no Query Parameters are specified and no proxy plugins should be executed.

```
{
        "adapter"  : "<adapter-identifier>",
        "time" : {
                    "from" : "1400250000",
                  },
        "queryParams" : { },
        "plugins" : []
}
```

Example 1 Example of a request

2.1.2.4 Example response

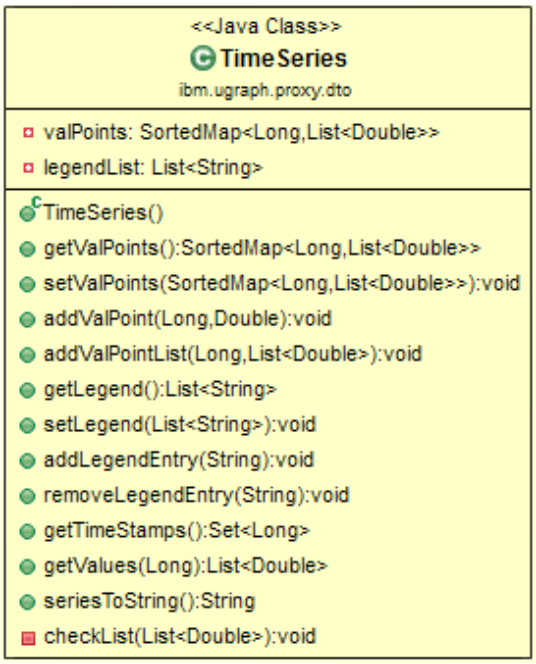
Example 2 is a an example of the data format that is returned. In the "data" attribute a time series with two time values and two graphs are returned. In the legend are the descriptions for each single graph. In this case "device1" and "device2".

```
{
       "data" : [[1386072000, 1527415.0, 1527417.0],[1386086400, 1590543.0, 15905.0]],
       "legend" : [ "x", "device1", "device2"]
}
```

Example 2 Example of a response

3 TimeSeries Object

The TimeSeries Object is used by the DataAdapter developer and the ProxyPlugin developer. The time series object provides a standardized way for the DataAdapter to store and interfacing time series data.



3.1 Members

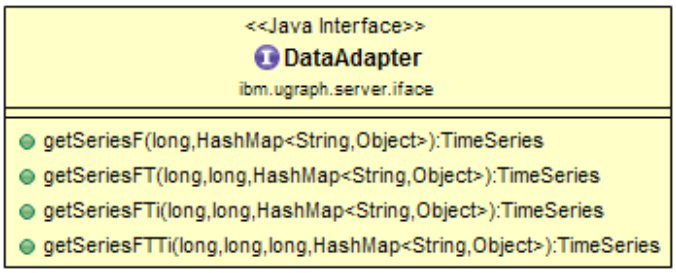
Name	Description
valPoints	SortedMap (sorted). Key is the timestamp, whereas the value is a List of Doubles that stores the values for that specific timestamp.
legendList	List of string that represents the legend of the graph.

3.2 Functions

Name	Description
getValPoints()	Returns the member valPoints.
setValPoints(TreeMap<Long, List<Double>>)	Sets the member valPoints.
addValPoint(Long, Double)	Takes a Long (utc timestamp) and a Double (value) and adds the ValuePoint to the member valPoints. If the timestamp doesn't exist in the TreeMap a new entry is automatically created.
addValPointList(Long, List<Double>)	Takes a Long (utc timestamp) and a List<Double> and adds the ValuePoints to the member valPoints. If the timestamp doesn't exist in the TreeMap a new entry is automatically created.
getLegend()	Return the member legend.
setLegend(List<String>)	Sets the member legend.
addLegendEntry(String)	Takes a string as parameter and adds it to the member legend.
removeLegendEntry(String)	Takes a string as parameter and removes the first matching entry. If no entry matching the string has been found the function is being quitted without any changes.
getTimeStamps()	Returns a Set<Long> containing all timestamps of the member valPoints.
getValues(Long)	Takes a Long (utc timestamp). Returns a List<Double> containing the values to the matching timestamp.
seriesToString()	Transforms the TimeSeries Object to a string in format of a JSON object, matching the format needed by the client. While parsing the data the size of the legendList has to be as big as the size of each List of values in the valPoints. In case of variation a TimeSeriesWrongFormatException is thrown.

4 DataAdapter Interface

The DataAdapter interface has to be implemented by the class that serves as DataAdapter. Four functions have to be implemented. For more information it is raccommended to read the Solution documentation.



4.1 Functions

Name	Description
getSeriesF(long, HashMap<String, Object>)	Takes a long (from – as a utc timestamp) and a HashMap containing DataAdapter specific parameters. The function returns the created TimeSeries Object.
getSeriesFT(long, long, HashMap<String, Object>)	Takes two longs (• from – as a utc timestamp • to – also a utc timestamp And a HashMap containing DataAdapter specific parameters. The function returns the created TimeSeries Object.
getSeriesFTi(long, long, HashMap<String, Object>)	Takes two longs • from – as a utc timestamp • the ticksize – a number that may vary for each Time Series service And a HashMap containing DataAdapter specific parameters. The function returns the created TimeSeries Object.
getSeriesFTTi(long, long, long, HashMap<String, Object>)	Takes three longs: • from – as a utc timestamp • to – also as utc timestamp • the ticksize – a number that may vary for each Time Series service And a HashMap containing DataAdapter specific parameters. The function returns the created TimeSeries Object.

4.2 Example implementation

The following code shows an example of a DataAdapter. As can be seen in the code the needed query parameters are

- token

```
package ibm.ugraph.zimon.impl;

import java.util.HashMap;
import java.util.Map;

import ibm.ugraph.server.dto.TimeSeries;
import ibm.ugraph.server.iface.DataAdapter;

import time.series.service.APIAccessor;

public class ExampleAdapter implements DataAdapter {

	/* The APIAccessor is the Class for Accessing the Time Series Service
	 * It may be, that the Time Series Service has to be accessed by calling
	 * a Web service. To keep the example easy we assume the APIAccessor
	 * has already functions to access the Time Series Data from the Time Series
	 * Service
	 */
	private APIAccessor accessor = new APIAccessor();
	GSON

	@Override
	public TimeSeries getSeriesF(long from, HashMap<String, Object> queryParams) throws Exception {
		/*
		 *  As Parameter the client sends the token, needed for the
		 *  authentication to the Time Series
		 */
		String token = retrieveParams(queryParams);
		/*
		 * The accessor takes as first parameter the token.
		 * The second parameter is the time declaration.
		 * We assume the request() Method of the API Accessor
		 * returns a JSON String which is serialized with GSON.
		 */
		String jsonResponse = accessor.request(token, from);
		Gson gson = new Gson();
		Query qry = gson.fromJson(jsonResponse, Query.class);
		TimeSeries ts = new TimeSeries();
		/*
		 * Again, for keeping the example understandable it is assumed
		 * that the accessor returns the Data right as needed.
		 */
		ts.setLegend(qry.getLegend());
		ts.setValuePoints(qry.getValuePoints());
		return ts;
	}
}
```

```
	@Override
	public TimeSeries getSeriesFT(long from, long to, HashMap<String, Object> queryParams) throws
Exception {
		String token = retrieveParams(queryParams);
		String jsonResponse = accessor.request(token, from, to);
		Gson gson = new Gson();
		Query qry = gson.fromJson(jsonResponse, Query.class);
		TimeSeries ts = new TimeSeries();
		ts.setLegend(qry.getLegend());
		ts.setValuePoints(qry.getValuePoints());
		return ts;
	}

	@Override
	public TimeSeries getSeriesFTi(long from, long ticksize, HashMap<String, Object> queryParams)
throws Exception {
		String token = retrieveParams(queryParams);
		String jsonResponse = accessor.request(token, from, ticksize);
		Gson gson = new Gson();
		Query qry = gson.fromJson(jsonResponse, Query.class);
		TimeSeries ts = new TimeSeries();
		ts.setLegend(qry.getLegend());
		ts.setValuePoints(qry.getValuePoints());
		return ts;
	}

	@Override
	public TimeSeries getSeriesFTTi(long from, long to, long ticksize, HashMap<String, Object>
queryParams)
			throws Exception {
		String token = retrieveParams(queryParams);
		String jsonResponse = accessor.request(token, from, to, ticksize);
		Gson gson = new Gson();
		Query qry = gson.fromJson(jsonResponse, Query.class);
		TimeSeries ts = new TimeSeries();
		ts.setLegend(qry.getLegend());
		ts.setValuePoints(qry.getValuePoints());
		return ts;
	}

	/*
	 * The function retrieves the token. If no token is contained
	 * in the HashMap a ArgumentMissingException is raised
	 */
	private String retrieveParams(HashMap<String, Object> queryParams) throws ArgumentMissingException
{
		if (queryParams.get("token") == null) {
			throw new ArgumentNotValidException();
		}
		return (String) queryParams.get("token");
	}
}
```

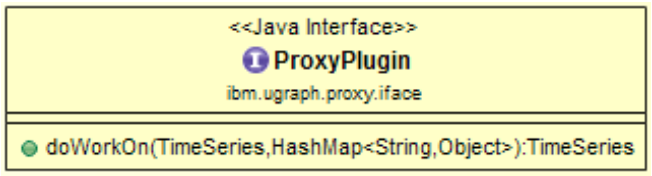
4.3 Example request body (relevant for client)

Assuming that the adapter identifier is known, the request from client side would look some like the following example.

```
{
        "adapter"  : "<adapter-identifier>",
        "time" : {
                  "from" : "1400250000",
                  "ticksize" : "1800"
                },
        "queryParams" : {
                                    "server" : "<server.url.or.ip>",
                                    "metrics" : [ "metric1", "metric2" ],
                                    "filters" : {
                                                 "filter1" : "filter",
                                                 "filter2" : "filter",
                                                },
                                    "useMetrics" : false
                                },
        "plugins" : [ ]
}
```

5 ProxyPlugin Interface

The ProxyPlugin interface has to be implemented by the class that serves as ProxyPlugin. Only one functions has to be implemented which is responsible for the manipulation of an existing TimeSeries Object.



5.1 Functions

Name	Description
doWorkOn(TimeSeries, HashMap<String, Object>	Takes a TimeSeries Object which contains data and a HashMap containing the specific parameters for the plugin. The function works directly on the passed TimeSeries Object and returns it.

5.2 Example implementation

This plugin calculates the average value per date and adds the calculated average to the TimeSeries-Object including the description in the legend. The name of the legend entry is passed as a plugin parameter by the client. In this case, the parameter that has to be passed is called "description". The name of such parameter has to be defined by the plugin developer and should always be used for the request with that plugin or a NullPointerException will be thrown and the client would be informed.

```
package ibm.ugraph.plugin.impl;

import java.util.HashMap;

import ibm.ugraph.server.dto.TimeSeries;
import ibm.ugraph.server.iface.ServerPlugin;

public class AveragePlugin implements ServerPlugin {

	@Override
	public TimeSeries doWorkOn(TimeSeries ts, HashMap<String, Object>
pluginParams) throws Exception {
		ts.addLegendEntry((String)pluginParams.get("description"));
		for (Long timestamp : ts.getTimeStamps()) {
			Double ave = 0.0;
			for (Double d : ts.getValues(timestamp)) {
				if (d != null) {
					ave += d;
				}
				continue;
			}
			ave /= ts.getValues(timestamp).size();
			ts.addValPoint(timestamp, ave);
		}
		return ts;
	}
}
```

5.3 Example request body (relevant for client)

Assuming that the pluginId is "average" the request by the client would look some like the following example.

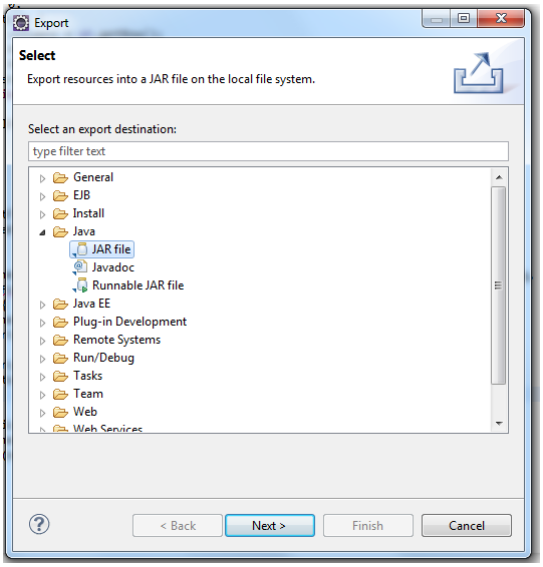
```
{
	"adapter"  : "<adapter-identifier>",
	"time" : {
			"from" : "1400250000"
		},
	"queryParams" : {},
	"plugins" : [ {
				"pluginId" : "average"
				"pluginParams" : {
						"description" : "Server Average"
					}
		} ]
}
```

6 Adding a new DataAdapter/ProxyPlugin

Due to the use of the Spring Framework it is easy to add additional DataAdapters and/or ProxyPlugins to a given proxy. Adding a DataAdapter/ProxyPlugin takes 4 steps.

6.1 Step 1: Export the implementation as a Jar

Export the project containing the implementation of the DataAdapter/ProxyPlugin as a jar



6.2 Step 2: Add the Jar to the proxy

Copy the exported jar file into the deployed proxy:

- <Path-to-TomCat7>/webapps/ugraph/WEB-INF/lib

6.3 Step 3: Add identifier to proxy Application Context

Open the application-config.xml file placed in

- <Path-to-TomCat7>/webapps/ugraph/WEB-INF/classes/spring

A new Spring bean has to be defined in this file for example lets add a DataAdapter and a ProxyPlugin whereby the ProxyPlugin id is "average" and the DataAdapter id is "adapter-identifier".

```
<beans>
	<!-- Adapters -->
	<bean id="adapter-identifier"
	class="package.name.of.the.implementation.ImplementedClass" lazy-
	init="true" primary="true" />
	<!-- Plugins -->
	<bean id="average"
	class="package.name.of.the.implementation.ImplementedClass" lazy-
	init="true" primary="true" />
</beans>
```

6.4 Step 4: Restart the proxy

As a last step the server has to be restarted.

7 Glossary

Time Series Service	A Database or web service which offers an interface for requesting Time Series Data
ValuePoint	Single value associated to a timestamp.