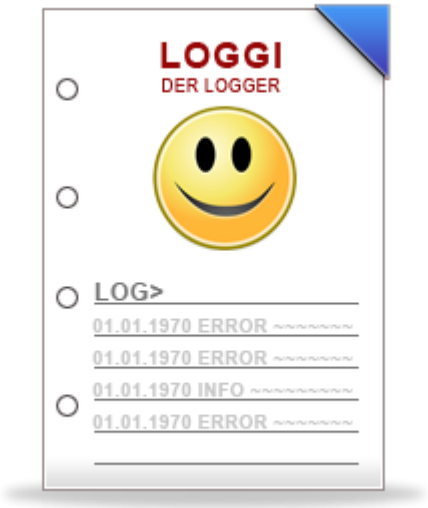


System-Spezifikation

Hochschule Luzern
Technik & Architektur

Software Komponenten – FS13

Gruppe 03 | Horw, 28.05.2013

Bontekoe Christian | Estermann Michael | Moor Simon | Rohrer Felix

Autoren

Bontekoe Christian	
Studiengang	Informatiker (Berufsbegleitend)
Adresse	
Telefon	
E-Mail	

Estermann Michael	
Studiengang	Informatiker (Berufsbegleitend)
Adresse	
Telefon	
E-Mail	

Moor Simon	
Studiengang	Informatiker (Berufsbegleitend)
Adresse	
Telefon	
E-Mail	

Rohrer Felix	
Studiengang	Informatiker (Berufsbegleitend)
Adresse	
Telefon	
E-Mail	

Änderungskontrolle

Version	Datum	Autor	Beschreibung
1.0	15.03.2013	Felix Rohrer	Vorlage erstellen, erste Texte schreiben
1.1	21.03.2012	Michael Estermann	Systemübersicht
1.2	13.04.2013	Felix Rohrer	Package Übersicht, LoggerServer, Nichtfunktionale Anforderungen ergänzt
1.3	16.04.2013	Michael Estermann	Klassendiagramme ergänzt, Sequenzdiagramm hinzugefügt, Funktionale Anforderungen definiert
--	16.04.2013	Christian Bontekoe	Version 1.3 freigegeben
1.4	19.04.2013	Michael Estermann	Schnittstellen spezifiziert, Review Fehler behoben (Siehe Review Dokument)
--	19.04.2013	Christian Bontekoe	Version 1.4 freigegeben
1.5	25.04.2013	Felix Rohrer	Softwaredesign, Schnittstellen ergänzen
--	25.04.2013	Christian Bontekoe	Version 1.5 freigegeben
1.6	19.05.2013	Felix Rohrer	Corba, Klassendiagramme ergänzt
2.0	24.05.2013	Christian Bontekoe	Version 2.0 freigegeben
2.1	27.05.2013	Felix Rohrer	Diverse Ergänzungen, Vervollständigungen
2.2	28.05.2013	Felix Rohrer	Anhänge eingefügt

Inhalt

1 Einführung

1.1 Systemübersicht

In diesem Projekt wird ein komponentenbasierendes Message Logger System entwickelt, welches aus vier Komponenten besteht.

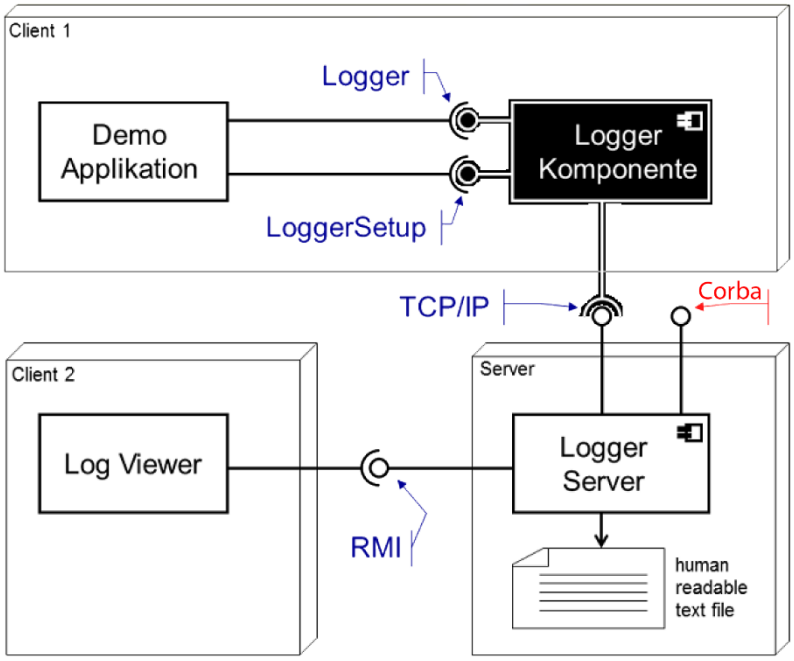


Abb. 1: Logger System gemäss Aufgabenstellung, inkl. Ergänzung Corba Interface

Logger Komponente: Die Logger Komponente empfängt von der Demo Applikation Log Messages. Diese Messages werden nach Empfang weitergeleitet an den Logger Server. Die Kommunikation erfolgt über TCP/IP in einem binären Format. In der zweiten Iteration gibt es zusätzlich noch eine Corba Schnittstelle zwischen dem LoggerServer und Logger Komponente. Weiter bietet die Logger Komponente über ihre Schnittstelle die Möglichkeit, sich von der Demoapplikation einstellen zu lassen (Logger Setup).

Logger Server: Der Logger Server ist die zentrale Stelle im ganzen System. Er empfängt von den einzelnen Logger Komponenten die Log Messages. Diese werden in einer lesbaren CSV-Datei abgelegt. Diese Logmessages werden weiter gesendet an alle Log Viewer, welche sich über RMI mit dem Server verbunden haben und die Log Messages angefordert haben.

Logger Viewer: Die Log Viewer Komponente zeigt vom Logger Server empfangenen Log Messages an. Diese Komponente wird möglichst minimal gehalten.

Demo Client: Diese Applikation bindet die Logger Komponente ein. Über diese Logger Komponente werden ihre Log Messages an den Server weitergesendet.

2 Architektur-Beschreibung

2.1 Funktionale Sicht

2.1.1 Logger Komponente

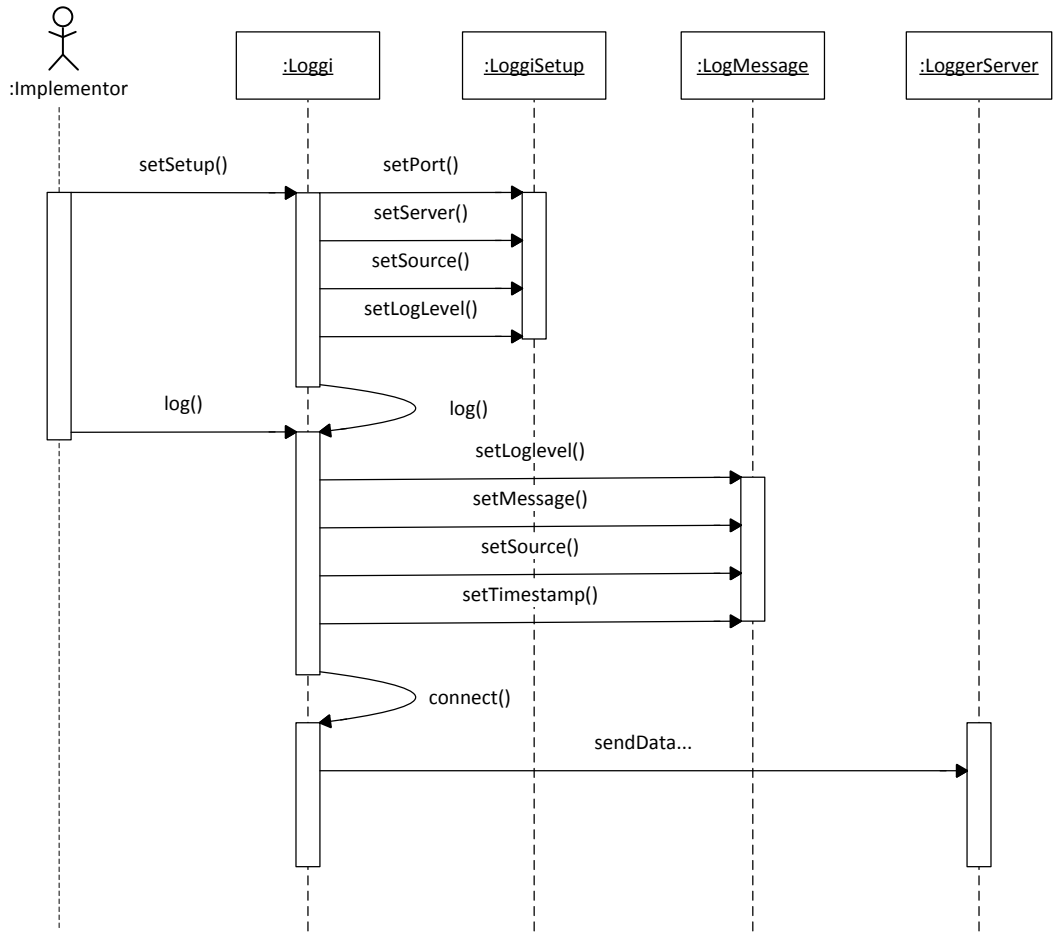


Abb. 2: Funktionale Sicht (Sequenzdiagramm)

2.2 Schnittstellenbeschreibung

2.2.1 Server ⇔ Viewer (RMI)

Zwischen Server und Viewer wird RMI eingesetzt. Der Viewer holt sich jeweils die Log-Messages, die seit seinem letzten Aufruf neu dazugekommen sind. Dafür speichert sich der Viewer eine Variable mit der ID der letzten Message.

Grundsätzlich würde sich hier das Observer-Pattern anbieten. Da dies nicht explizit gefordert wurde und es für unser Projekt nicht notwendig war, haben wir ein „Polling-Verfahren“ ohne Observer-Pattern gewählt.

Das RMI-Interface ist im Package logger.common (ch.hslu.swk.g03.logger.common) definiert und heisst **ServerRmiInterface**. Beim Logger.Server (ch.hslu.swk.g03.logger.server) ist das Interface in der Klasse **ServerRMI** implementiert.

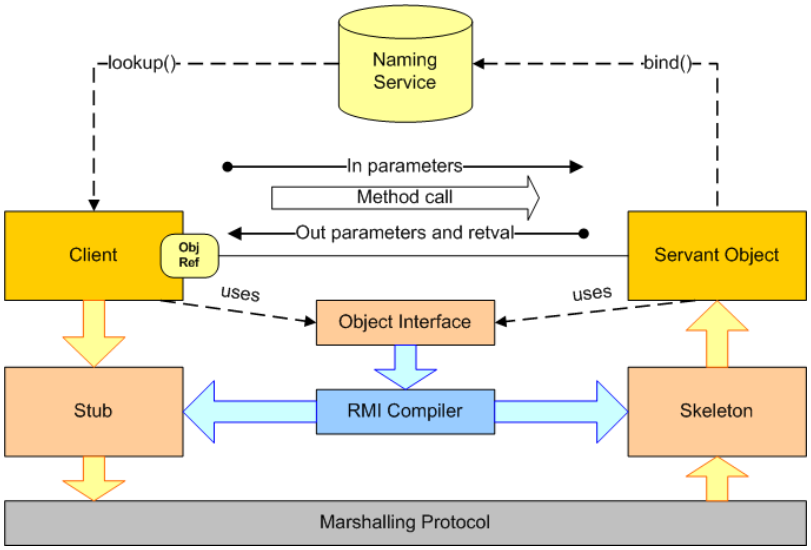


Abb. 3: RMI Funktionalität (Quelle: www.xatlantis.ch)

Abb. 4: ServerRmiInterface

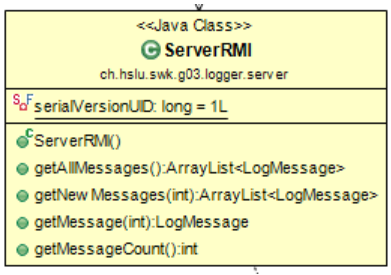


Abb. 5: ServerRMI (Rmi Interface impl.)

Die Einstellungen vom Viewer werden mithilfe der Datei viewer.properties gesteuert.

```
#HSLU T&A -- SWK.F13.Gruppe03 -- LoggerViewer Properties
RMIServerHost = 10.3.98.106
RMIServerPort = 10126
RMIServerName = G03RMIServer
```

Abb. 6:viewer.properties

2.2.2 Server ⇔ Komponente (TCP/IP, Corba)

Als Schnittstelle zwischen dem Server und der Komponente wurde in Version 1 eine TCP/IP Kommunikation vorgegeben. Übermittelt werden dazu TCP Pakete, welche serialisierte Java Objekte beinhalten. Dazu sendet der Client vor jedem Objekt ein int32 Wert, damit der Server weiss, wie viele Daten er einlesen muss, bevor er die Verbindung wieder schliessen darf.

Um die Objekte zu serialisieren wurde eine Utils Klasse **ConverterUtils** implementiert (Package: logger.common), welche Objekte in ein byte Array und wieder zurück konvertieren kann.

In Version 1 wurde für die Übermittlung die eigene Klasse **LogMessage** definiert (Package: logger.common).

In Version 2 wurde das CORBA-Interface/Schnittstelle, welches ebenfalls vom Schnittstellenteam definiert wurde, verwendet.

Die Komponente implementiert dieses in der Klasse **LoggiCorba** (Package: logger.component).

Die Kommunikation wird mithilfe der Jacorb Bibliotheken sowie die generierten IDL Hilfsklassen ermöglicht.

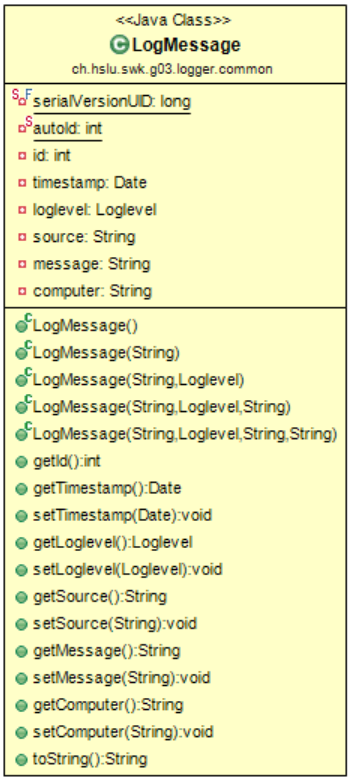


Abb. 7: LogMessage

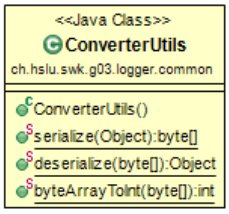


Abb. 8: ConverterUtils

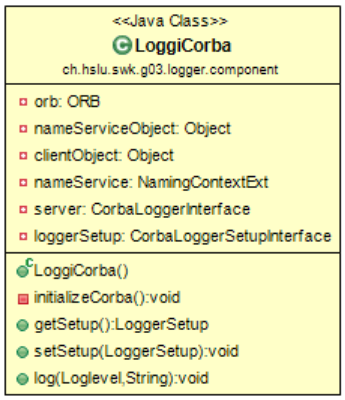


Abb. 9: LoggiCorba

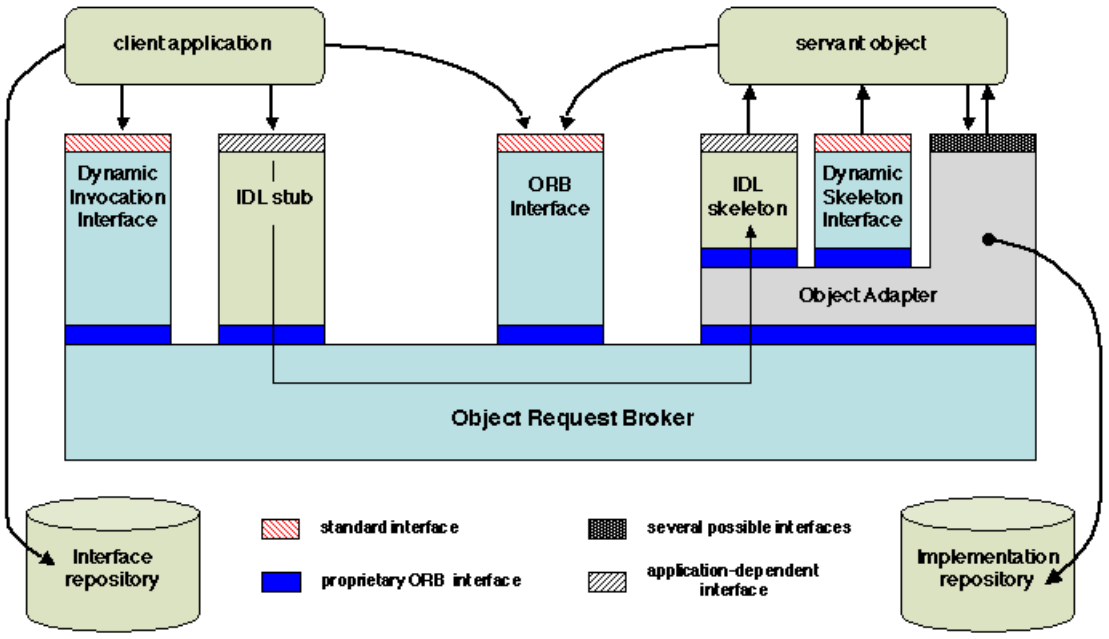


Abb. 10: Corba Teilsysteme (Quelle: http://proton.inrialpes.fr/~krakowia/)

Das Corba-Interface, resp. das dazugehörige IDL sieht wie folgt aus:

```
module corbalogger {
    enum CorbaLogLevel {
        OFF,
        ERROR,
        WARNING,
        INFO,
        DEBUG
    };

    interface CorbaLoggerSetupInterface {
        void setCorbaLogMessageSource(in string corbaLogMessageSource);

        string getCorbaLogMessageSource();

        void setCorbaLogLevel(in CorbaLogLevel concreteCorbaLogLevel);

        CorbaLogLevel getCorbaLogLevel();
    };

    interface CorbaLoggerInterface {
        exception CorbaLoggerCouldNotLogException {
        };

        void setCorbaLoggerSetup(in CorbaLoggerSetupInterface concreteCorbaLoggerSetup);

        CorbaLoggerSetupInterface getCorbaLoggerSetup();

        void corbaLog(in CorbaLogLevel concreteCorbaLogLevel, in string corbaLogMessage,
                      out boolean corbaLogSuccess) raises(CorbaLoggerCouldNotLogException);
    };
};
```

Abb. 11: Corba LoggerInterface.idl

In der Klasse **ServerCorba** (Package logger.server) wird ORB instanziiert. Ebenfalls Erzeugen und Binden der Interface Servant.

In der Klasse **ConcreteCorbaLoggerInterfacePOA** (Package logger.server) ist die konkrete Implementierung des InterfacePOA (CorbaLoggerInterface).

In der Klasse **ConcreteCorbaLoggerSetupInterfacePOA** (Package: logger.server) ist die konkrete Implementierung des SetupInterfacePOA (CorbaLoggerSetupInterface).

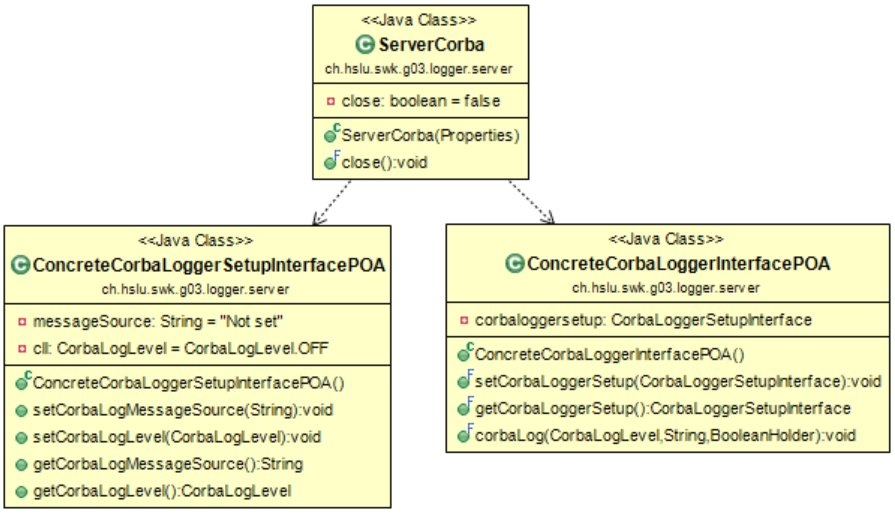


Abb. 12: Logger Server: Corba Interface Implementierung

Die Einstellungen vom LoggerServer werden mithilfe der Datei server.properties gesteuert.

```
#HSLU T&A -- SWK.F13.Gruppe03 -- LoggerServer Properties
#CSV-Logfile
LogFile = G03Logger.log

#TCP-Server
TCPPort = 10123

#RMI-Server
RMIServerName = G03RMIServer
RMIPort       = 10124

#Corba-Server
corba.orbclass          = org.jacorb.orb.ORB
corba.orbsingletonclass = org.jacorb.orb.ORBSingleton
corba.nameservice       = corbaloc::10.3.98.106:38693/NameService
corba.bindname          = G03CorbaServer
corba.bindnamesetup     = G03CorbaSetup
corba.properties        = C:\\JacORB\\etc\\jacorb.properties
```

Abb. 13:server.properties

Der Ablauf eines Corba Methodenaufrufs sieht wie folgt aus:

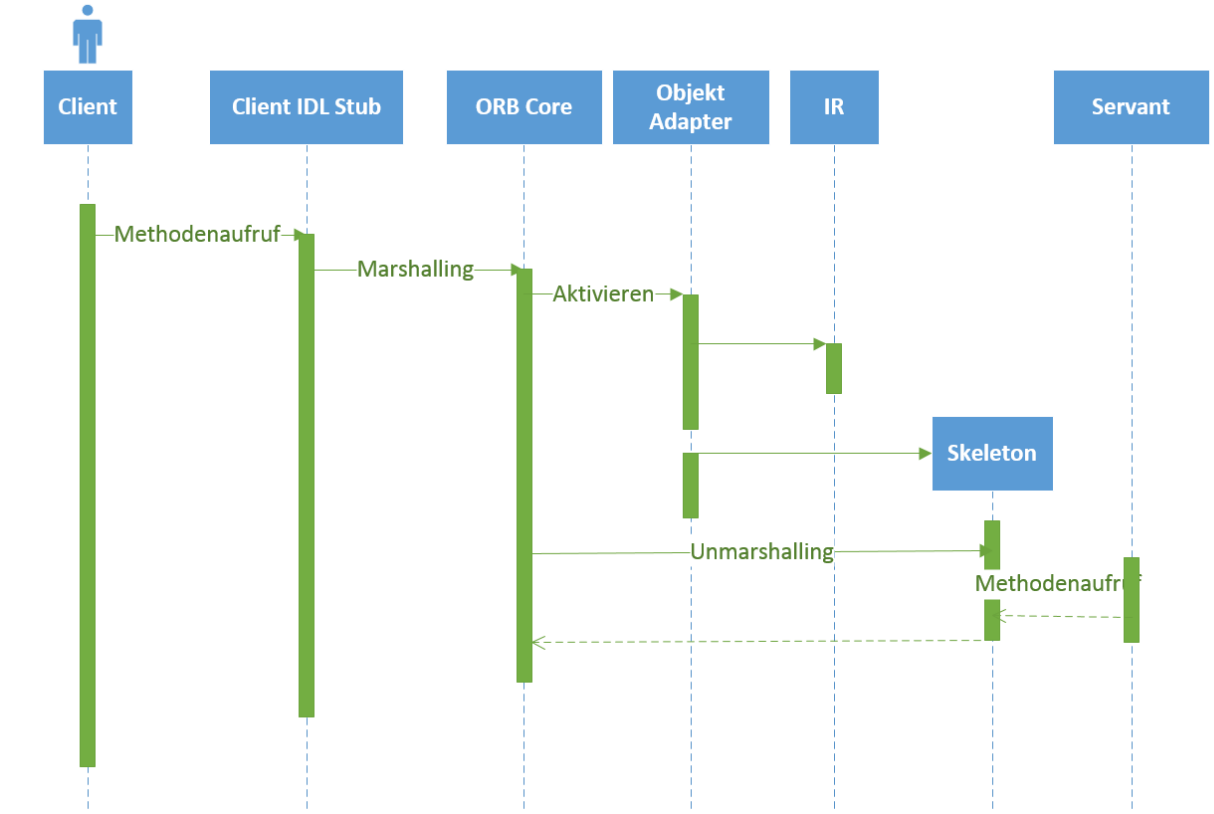


Abb. 14: Ablauf eines CORBA Methodenaufrufs (IR: Interface Repository)

2.2.3 Server ⇔ Logfile

In der Klasse **LogMessage** (Package: logger.common) wird die Methode toString() überschrieben, um eine LogMessage direkt im CSV speichern zu können.

Es wird das Standard CSV Format, welches in RFC 4180 definiert ist, verwendet. Entsprechend werden die Anführungszeichen " maskiert (escaped), dazu werden diese verdoppelt.

```
"ID","Timestamp","Loglevel","Computer","Source","Message"
3,"Mai 24,2013 14:01:15.707","INFO","Mobilstation01","DemoClient","Nr 1: This is an Info Mes-
sage [3]    #2013-05-24 14:01:15.687#"
4,"Mai 24,2013 14:01:15.917","WARNING","Mobilstation01","DemoClient","Nr 2: This is a Warning
Message [2]  #2013-05-24 14:01:15.917#"
5,"Mai 24,2013 14:01:16.117","DEBUG","Mobilstation01","DemoClient","Nr 3: This is a Debug
Message [4]    #2013-05-24 14:01:16.117#"
6,"Mai 24,2013 14:01:16.317","DEBUG","Mobilstation01","DemoClient","Nr 4: This is a Debug
Message [4]    #2013-05-24 14:01:16.317#"
7,"Mai 24,2013 14:01:16.517","ERROR","Mobilstation01","DemoClient","Nr 5: This is an Error
Message [1]   #2013-05-24 14:01:16.517#"
8,"Mai 24,2013 14:01:16.797","DEBUG","Mobilstation01","DemoClient","Nr 6: This is a Debug
Message [4]    #2013-05-24 14:01:16.717#"
9,"Mai 24,2013 14:01:17.127","ERROR","Mobilstation01","DemoClient","Nr 7: This is an Error
Message [1]   #2013-05-24 14:01:17.007#"
10,"Mai 24,2013 14:01:17.427","ERROR","Mobilstation01","DemoClient","Nr 8: This is an Error
Message [1]   #2013-05-24 14:01:17.327#"
```

Abb. 15: Auszug aus dem CSV-Logfile für die persistente Speicherung.

2.2.4 Demo Client ⇔ Komponente

Der Demo Client kommuniziert mit der Komponente über ein vordefiniertes Interface, welches vom Interface Team bestimmt wurde. Die aktuelle Version ist unter https://dev.enterpriselab.ch/education/swk.f1301.00/ veröffentlich.

Das Corba Interface (IDL) wurde bereits in Kapitel 2.2.2 Server ⇔ Komponente (TCP/IP, Corba) erläutert.

Das Interface v1 hat folgende Klassenstruktur: (ohne Corba)

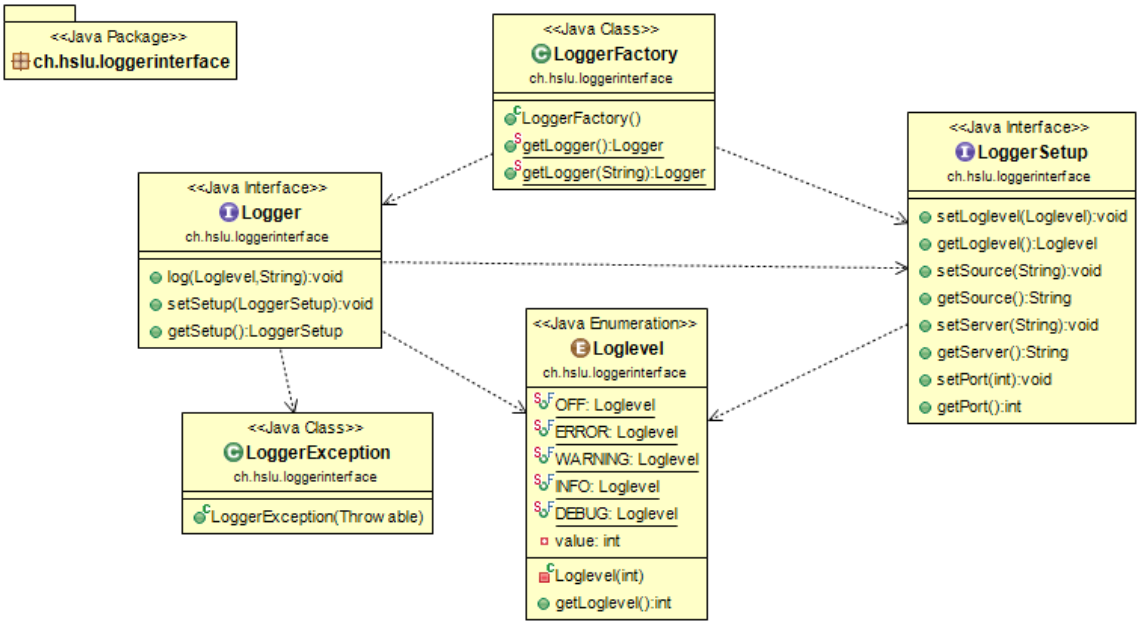


Abb. 16: LoggerInterface: Klassen Übersicht (ohne Corba)

Das CorbaInterface hat folgende Klassenstruktur: (ohne LoggerSetup, etc.)

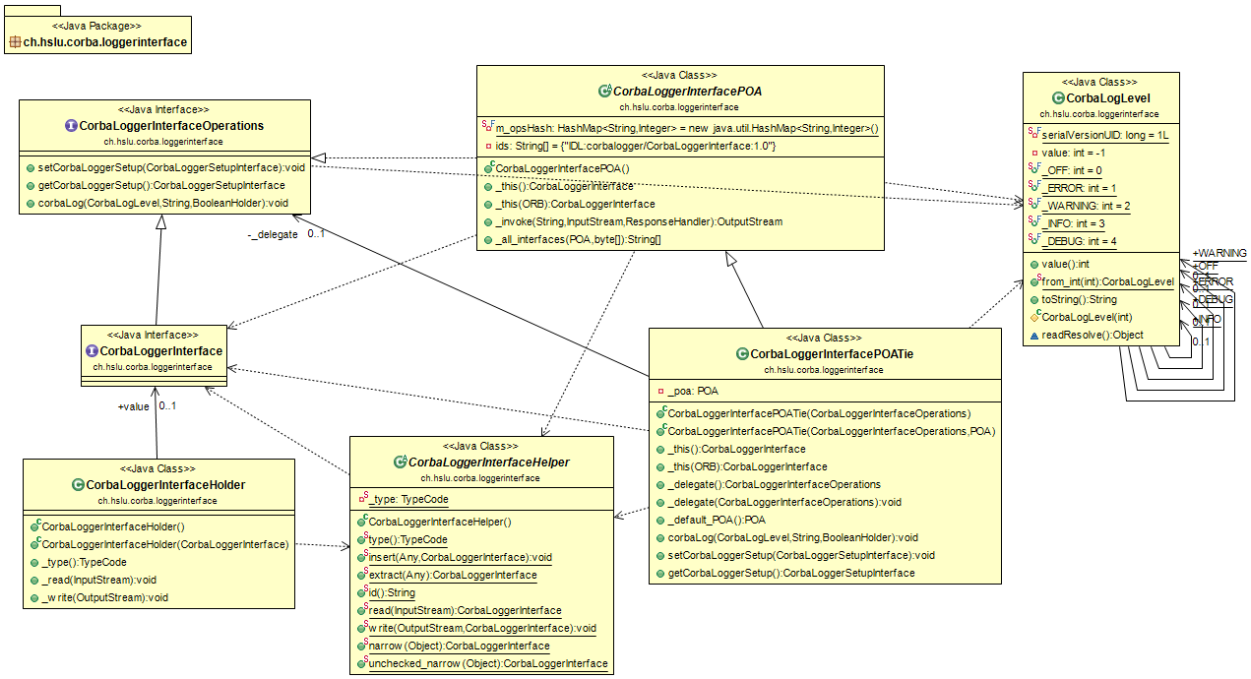


Abb. 17: CorbaLoggerInterface: Klassen Übersicht (Ausschnitt LoggerInterface ohne LoggerSetup)

Die Einstellungen vom LoggerDemoClient werden mithilfe der Datei logger.properties gesteuert.

```
#HSLU T&A -- SWK.F13.Gruppe03 – LoggerDemoClient Properties
#logger jar filename
logger_jar         = swk_logger_component-2.0.3.jar

#logger classname
logger_class       = ch.hslu.swk.g03.logger.component.LoggiCorba

#logger setup classname
logger_setup_class = ch.hslu.swk.g03.logger.component.LoggiSetup

#RMI-Server IP-Address
Server             = 10.3.98.106

#RMI-Server Port
Port               = 10123

#DemoAppClient SourceName
Source             = DemoClient
```

Abb. 18:logger.properties

Falls Corba verwendet werden möchte, können zusätzlich die Corba Einstellungen mithilfe der Datei corba.properties gesteuert werden.

```
#HSLU T&A -- SWK.F13.Gruppe03 – LoggerDemoClient Corba Properties
#CorbaServer Bindname
corba.bindname         = G03CorbaServer

#CorbaServer Bindanem Setup
corba.bindnamesetup    = G03CorbaSetup

#Corba NameService Location
ORBInitRef.NameService = corbaloc::10.3.98.106:38693/NameService

#Corba Jacorb Properiesfile
corba.properties       = C:\\JacORB\\etc\\jacorb.properties
```

Abb. 19:corba.properties

Das Properties-File muss im gleichen Verzeichnis liegen wie die JAR-Datei.

2.3 Entwicklungssicht

2.3.1 Packages Übersicht

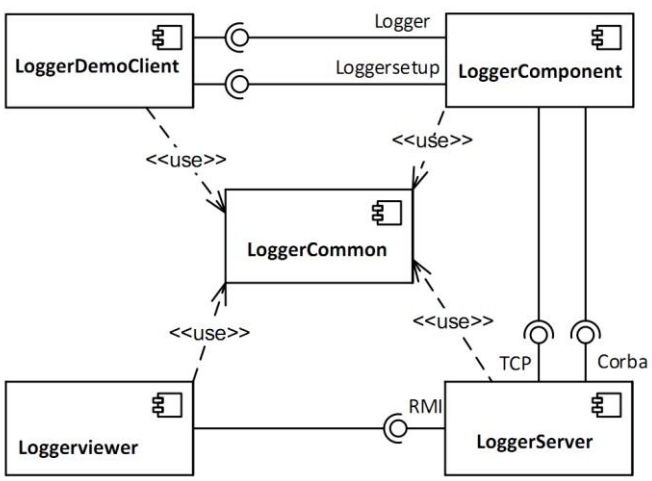


Abb. 20: Logger Übersicht mit Schnittstellen

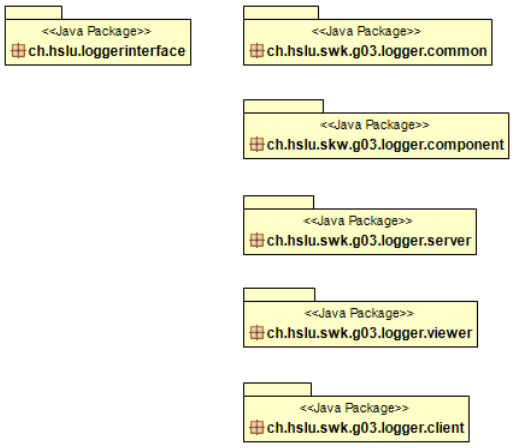


Abb. 21: Logger: Package Übersicht

2.3.2 Logger Common Klassenübersicht

In Logger Common ist die Klasse LogMessage für die Messages sowie das RMI Interface ServerRmiInterface definiert. Ebenfalls gibt es eine zusätzliche Klasse für ConverterUtils, primär um die LogMessage zwischen LoggerComponenten und LoggerServer zu serialisieren / deserialisieren.

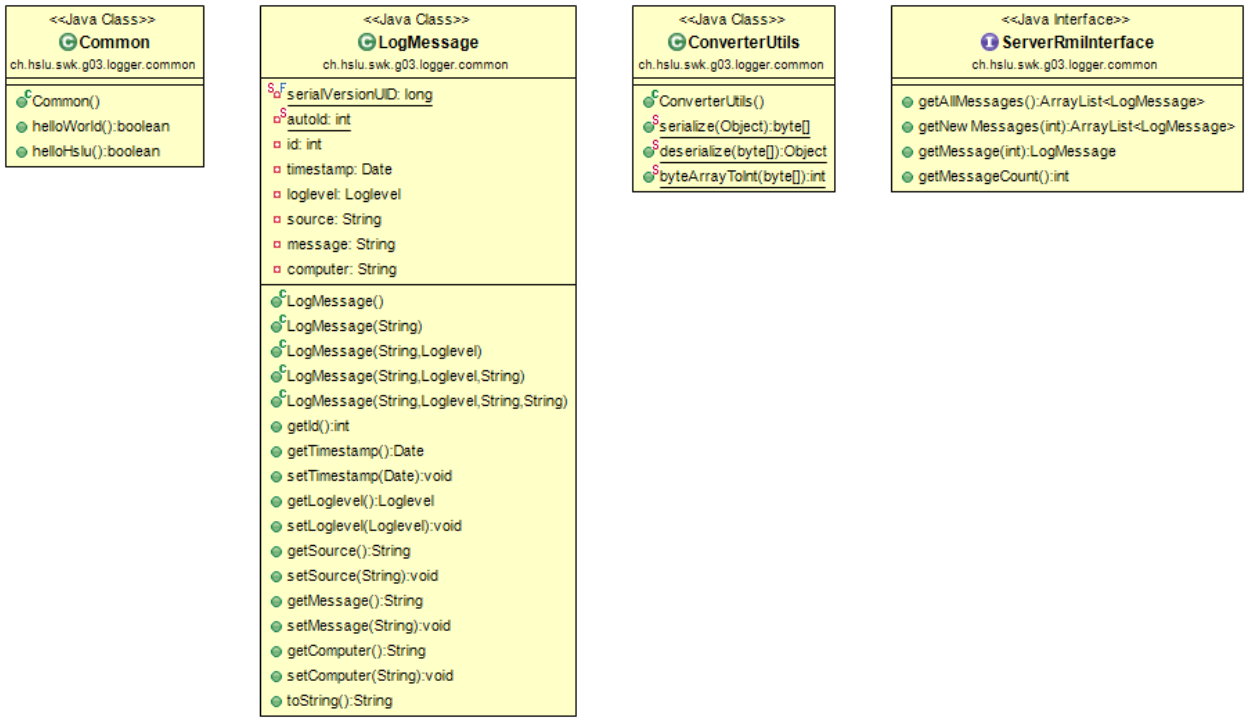


Abb. 22: Logger.Common: Klassen Übersicht

2.3.3 Logger Component Klassenübersicht

Die Loggerkomponente implementiert die Funktionalität des Loggerinterfaces welches vom Schnittstellen-Team definiert wurde. Sobald bei der Logger-komponente ein gültiges Setup gesetzt wurde, ist diese bereit Logmessages an einen Server über die TCP-Schnittstelle (Kapitel 2.2.2) zu senden.

Mit Release 2 kann alternativ das CORBA-Interface verwendet werden, welches auch vom Schnittstellen Team definiert wurde.

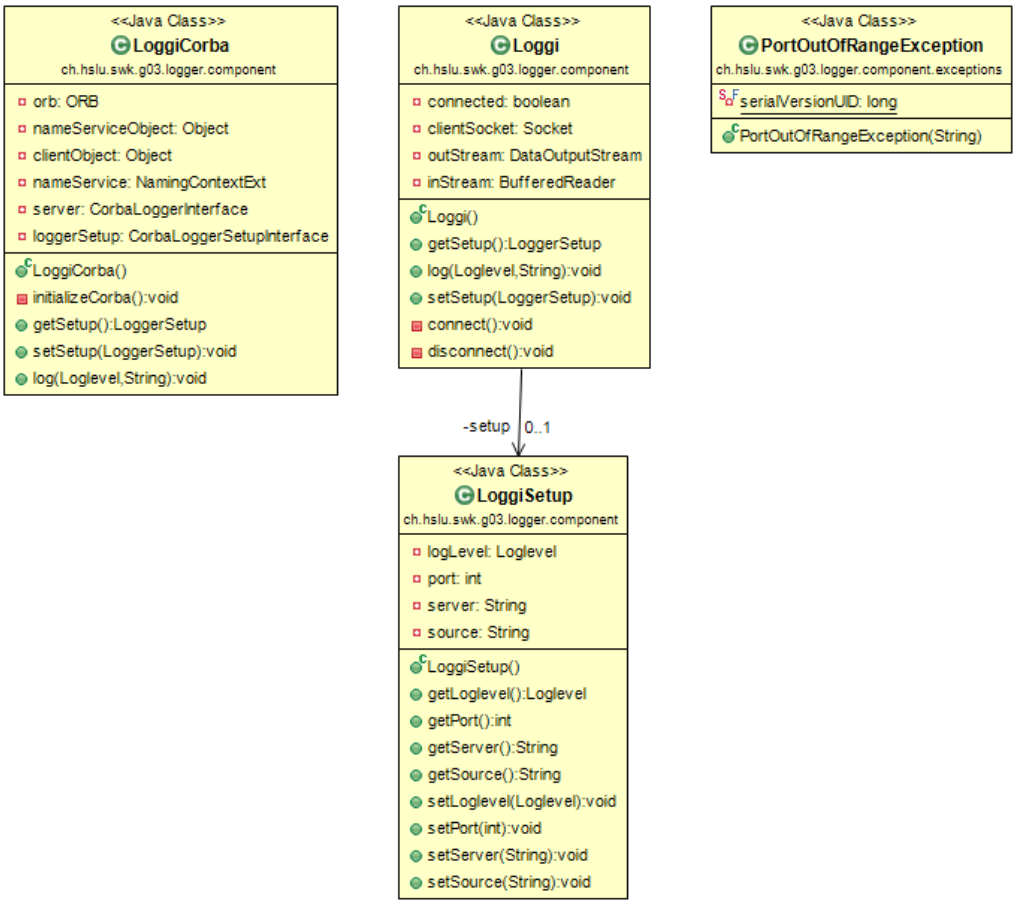
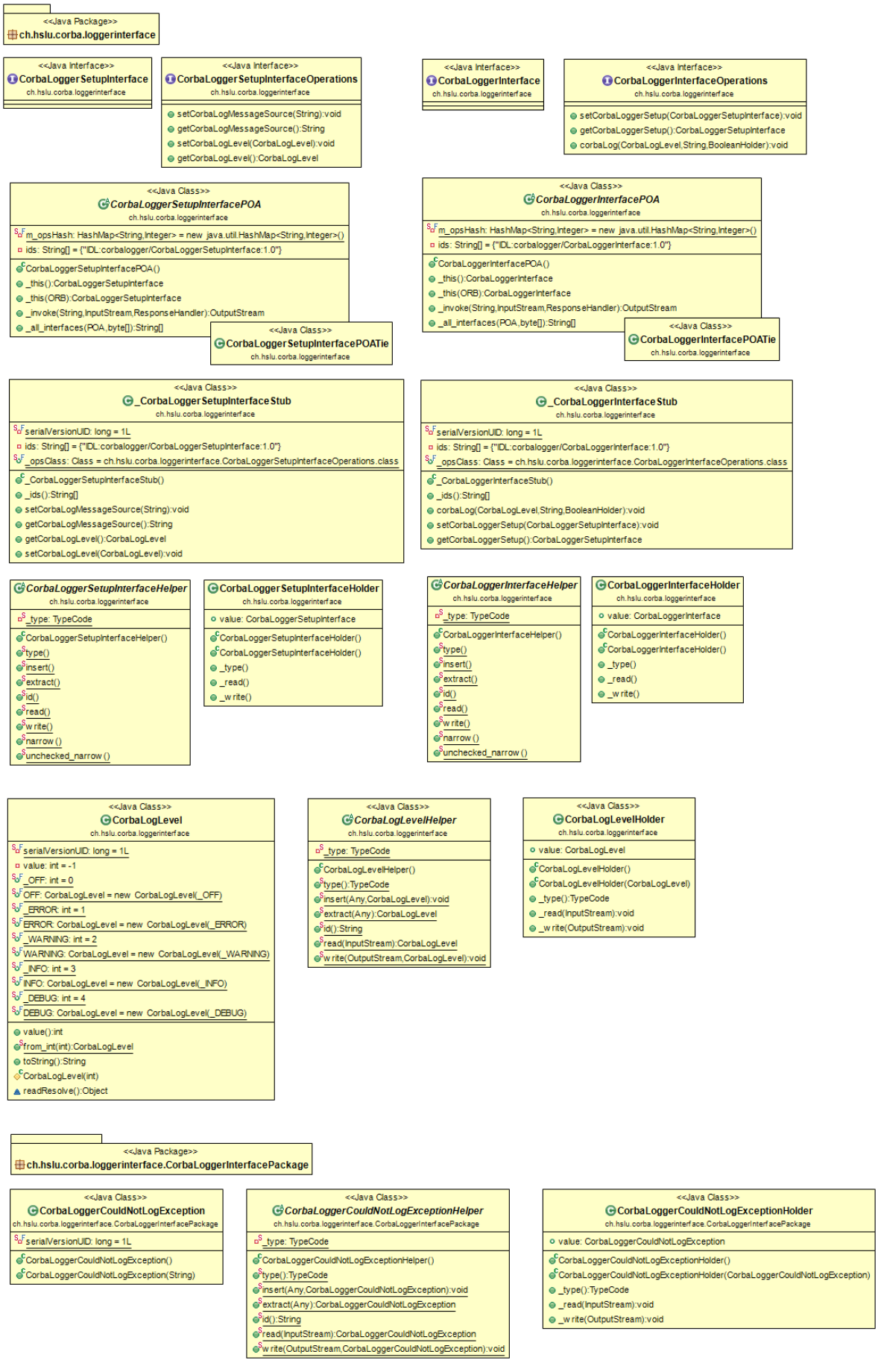


Abb. 23: Logger.Component: Klassen Übersicht

2.3.4 CorbaLogger Interface Klassenübersicht



2.3.5 Logger Server Klassenübersicht

Der LoggerServer implementiert die Grundfunktionalität für den Server. Zusätzlich gibt es die Klasse ServerTCP welcher als Singelton implementiert ist. Für jeden Client der sich mit dem Server verbindet wird einen einzelnen ServerTCPConnThread gestartet. Für RMI wird ein einzelner Thread ServerRMI ausgeführt. Dieses setzt entsprechend das ServerRMIInterface um. Die LoggerDB ist ebenfalls als Singelton implementiert.

Mit der zweiten Iteration ist zusätzlich noch eine Klasse für den Corba-Server dazu gekommen. Ebenfalls werden die zwei POA's ConcreteCorbaLoggerSetupInterfacePOA und ConcreteCorbaLoggerInterfacePOA implementiert.

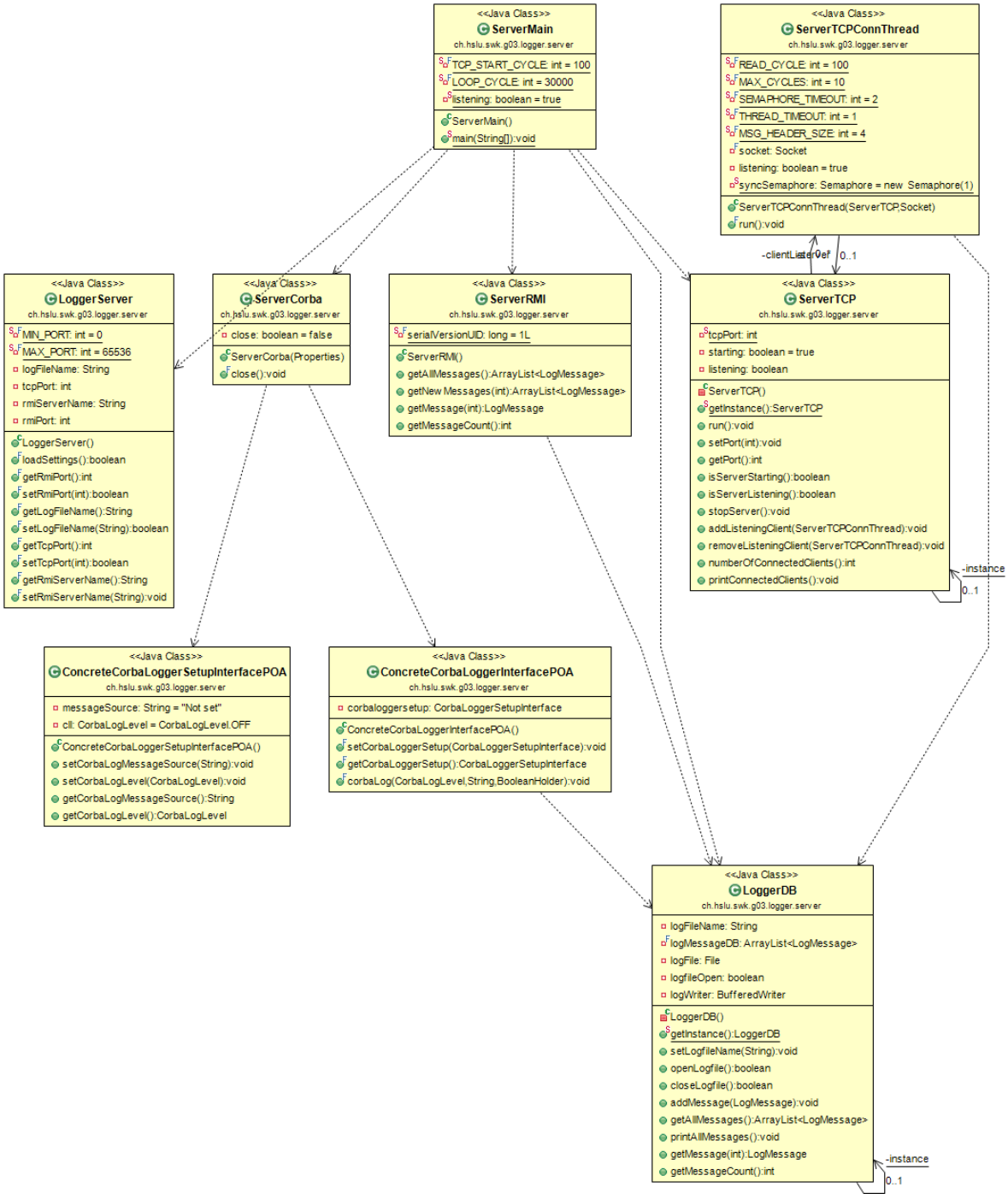


Abb. 24: Logger.Server: Klassen Übersicht

2.3.6 Logger Client Klassenübersicht

Der LoggerClient verwendet das vom Schnittstellen-Team definierte Interface. Somit kann die Logger-Komponente ausgetauscht werden, ohne dass der Client angepasst werden muss.

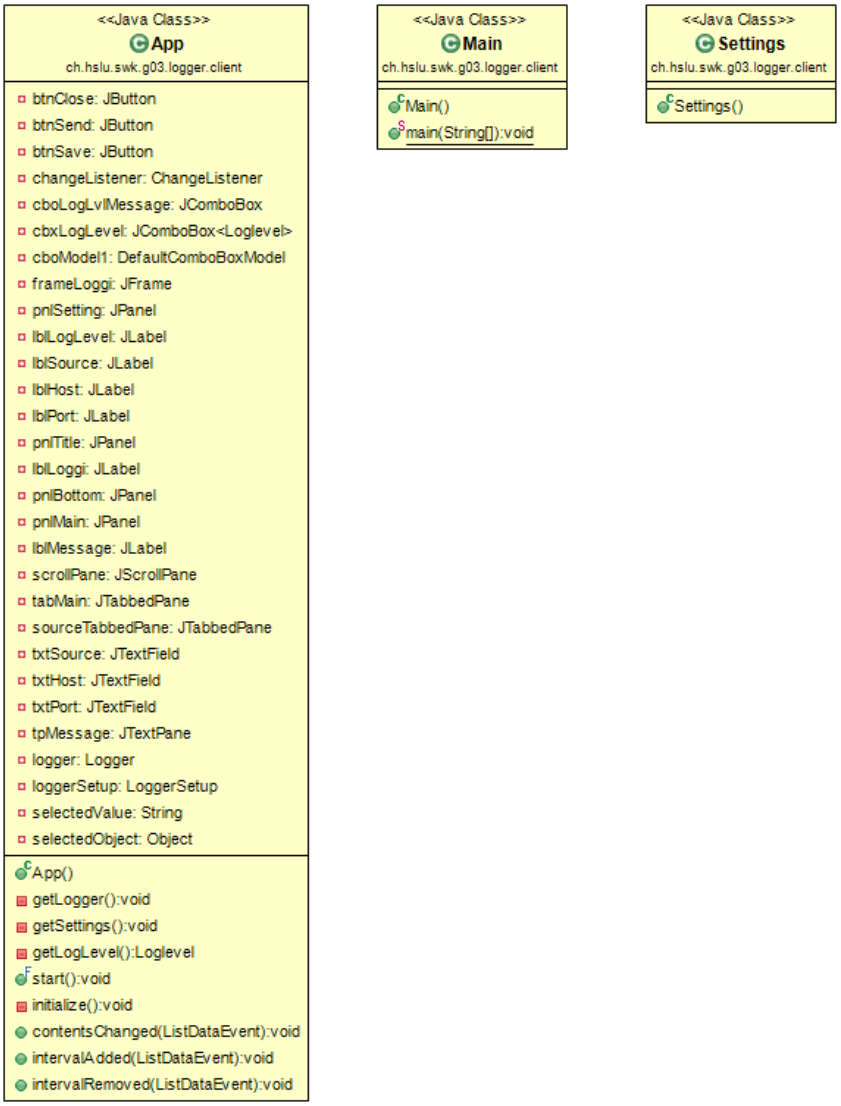


Abb. 25: Logger.Client: Klassen Übersicht

2.3.7 Logger Viewer Klassenübersicht

Das Loggergui zeigt die Messages an, die an den Server gesendet werden. Dazu holt der FetchThread via RMI mit dem Interface ServerRMIInterface dauernd die neuen Nachrichten vom Server und fügt diese im GUI ein.

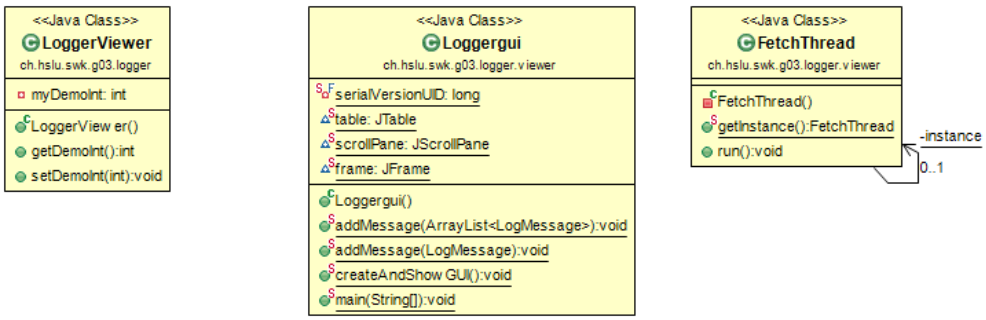


Abb. 26: Logger.Viewer: Klassen Übersicht

3 Softwareanforderungen und Softwaredesign

3.1 Softwareanforderungen

3.1.1 Nichtfunktionale Anforderungen

Anforderung	Beschreibung
Zuverlässigkeit	Es sollen keine Abstürze vorkommen
Benutzerfreundlichkeit	Intuitive und verständliche Oberfläche (GUI), Logfile sollte einfach lesbar sein.
Effizienz	Systemressourcen werden nicht unnötig ausgelastet
Performance	Logmessages werden ohne grosse Verzögerungen übermittelt und gespeichert.
Wartbarkeit	Nachträgliche und Änderungen sind ohne grosse Umstände möglich
Erweiterbarkeit	Nachträgliche Ergänzungen sind ohne grosse Umstände möglich
Korrektheit	Die Logmessages werden korrekt übermittelt.

3.1.2 Funktionale Anforderungen

Anforderung	Beschreibung
Austauschbarkeit I	Die Loggerkomponente muss den Schnittstellenanforderungen, welche durch das Interface-Team gestellt wurden, entsprechen. Dies ermöglicht es, die Loggerkomponente auszutauschen.
Austauschbarkeit II	Die Loggerkomponente muss innerhalb des Clients ohne jegliche Codeanpassungen austauschbar sein.
Datenkonsistenz	Die vom Loggerserver empfangenen Daten müssen korrekt in eine lesbare Datei geschrieben werden. Dabei müssen mindestens die Eigenschaften Quelle, Zeitstempel, Loglevel und Nachricht vorhanden sein.
Konfigurierbarkeit	Falls eine Komponente statische Eigenschaften hat, müssen diese über eine Datei konfigurierbar sein.
Filterbarkeit	Über die Schnittstelle zur Komponente muss die Applikation für jeden Logeintrag einen Loglevel mitsenden. Dieser Loglevel muss höher oder gleich sein, als der in der Komponenten definierte minimale Loglevel, dass die Logmessage an den Loggerserver gesendet wird. Andernfalls wird die Logmessage verworfen.
Kommunikation I	Die Kommunikation zwischen dem Loggerserver und dem Viewer muss mittels RMI implementiert werden.
Kommunikation II	Der Loggerserver und die Loggerkomponente müssen mittels TCP/IP kommunizieren. In welcher Form die Daten hierbei übermittelt werden ist nicht vorgeschrieben.
Mehrfachverbindungen	Der Loggerserver muss mittels Threading implementiert werden, so dass sich mehrere Loggerkomponenten gleichzeitig mit ihm verbinden können.

3.2 Softwaredesign

1. Nach Möglichkeit wird das Model-View-Controller (MVC) Architekturmuster angewendet.
2. Die Komponente und der Loggerserver müssen Threadsafe programmiert werden (synchronized, lock, semaphore)
3. Die Logmessages der Komponente werden mittels Java Objektserialisierung binär serialisiert und über die TCP/IP Verbindung an den Server gesendet. Mit Release 2 kann alternativ die CORBA-Schnittstelle verwendet werden.
4. Das Loggerinterface implementiert ein Factory-Pattern, welches ermöglicht, dass der Demo Client über das Loggerinterface eine Loggerkompontente instanzieren kann, welche vorgängig in einer Property-Datei definiert wurde.

4 Environment-Anforderungen

4.1 Hardware

Der Logger läuft auf fast jeder Hardware. Grundsätzlich muss Java lauffähig sein, an die Ressourcen werden vom Logger keine speziellen Anforderungen gestellt. Für die GUI-Komponenten wird OpenGL benötigt.

4.2 Software

Alle Betriebssysteme, welche Java 7 unterstützen.

4.3 Java Virtual Machine

Damit der Logger läuft, muss die JVM 7 installiert sein.

Abbildungsverzeichnis

Anhang

Projektauftrag: Message-Logger

Hochschule Luzern
Technik & Architektur

Modul: TA.BA_SWK.F1301

Projektauftrag: Message-Logger

M. Jud, R. Gisler, P. Sollberger

Version 1.0 / 20.02.2013

Hochschule Luzern
Technik & Architektur

Inhalt

Hochschule Luzern
Technik & Architektur

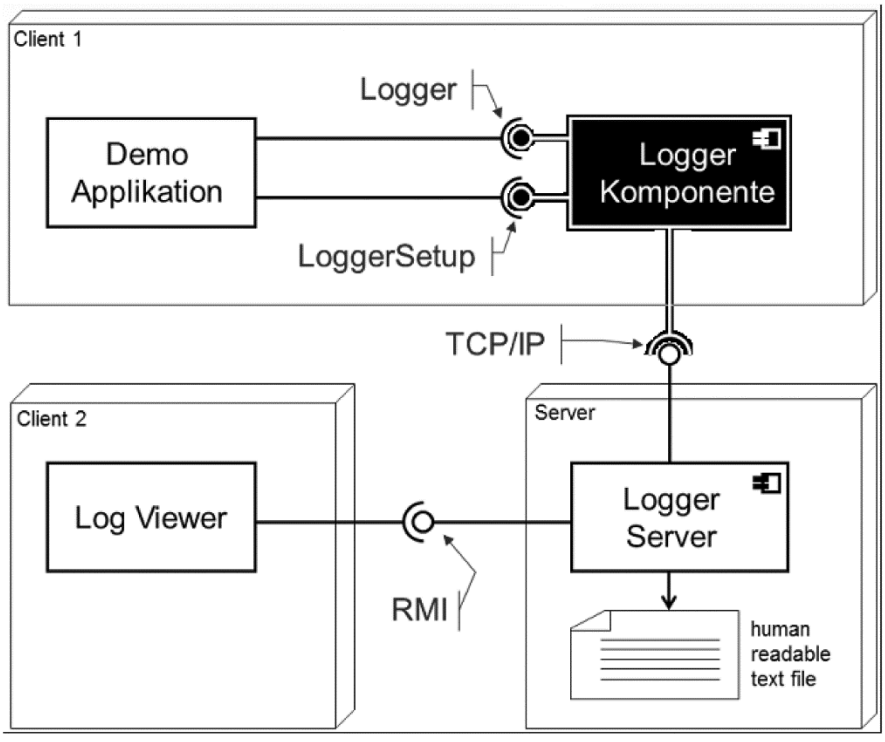
1. Einleitung

Verschiedene Betriebssysteme stellen integrierte Logging-Mechanismen zur Verfügung: Unix-basierende Systeme z.B. das so genannte 'Syslog' und Windows-basierende Systeme das 'Event-Log'. Im Rahmen dieser Aufgabe soll ein alternatives und komponentenbasierendes Message-Loggingsystem entwickelt werden, welches unabhängig von der Plattform und auch auf mehrere Hosts verteilt verwendet werden kann.

2. Anforderungen

2.1. Grundfunktionalität des Message-Loggers

Bei der **Logger-Komponente** handelt es sich um eine Software-Komponente, die sehr einfach in bestehende Java-Applikationen eingebunden werden kann. Eine Applikation kann durch einfache Methodenaufrufe textbasierende Meldungen (Messages) mit ihrer Logger-Komponente dauerhaft aufzeichnen. Die Meldungen aller Applikationen sollen in einem zentralen Logger Server in einem vordefinierten Format gespeichert werden.
Mit einem **Log-Viewer**, einer eigenständigen Anwendung, die unabhängig von der Logger-Komponente arbeitet, können die auf dem Server eintreffenden Messages (einer oder mehrerer Applikationen) online verfolgt und angezeigt werden. Online bedeutet, dass Meldungen sofort und ohne Benutzerinteraktion angezeigt werden, sobald eine neue Message auf dem Server eintrifft.



2.2. Einsatzgebiet

Das Message-Loggingsystem kann z.B. für Debug-Zwecke (Tracing, die Applikation meldet durch Meldungseinträge den zurückgelegten Programmfluss und Zwischenresultate) oder für das Error-Logging im Feld (die Applikation meldet wichtige Fehlersituationen und notwendige Information dazu) verwendet werden.

Hochschule Luzern
Technik & Architektur

2.3. Muss-Features

Nr.	Bezeichnung
1	Die Logger-Komponente muss als Komponente (mit Interface) realisiert werden.
2	Die dauerhafte Speicherung der Messages erfolgt in einem einfachen Textfile (welches somit auch mit ‚Bordmitteln' des jeweiligen Betriebssystems eingesehen werden kann). Das File enthält mindestens die Quelle der Logmeldung, einen Zeitstempel, den Message-Level und den Message-Text.
3	Die Logger-Komponente ist austauschbar und plattformunabhängig zu realisieren. Der Komponentenaustausch muss ausserhalb der Entwicklungsumgebung und **ohne** Code-Anpassung, d.h. ohne Neukompilation möglich sein.
4	Es muss möglich sein, dass mehrere Applikationen über ihre jeweilige Instanz der Logger-Komponente parallel auf den zentralen Message-Server loggen.
5	Log-Viewer: Unabhängige Anwendung, welche sowohl bereits früher aufgezeichnete Einträge als auch die aktuellen Einträge online anzeigt. Die Kommunikation soll dabei über RMI erfolgen (RMI = Remote Method Invocation).
6	Bei jedem Log-Eintrag hat die aufrufende Applikation einen **Message-Level** mitzugeben (z.B.: Fehler, wichtige Meldung, normale Meldung, unwichtige Meldung). Über die API der Logger-Komponente kann ein Level-Filter gesetzt werden. Damit kann definiert werden, welche Meldungen (mit welchem Level) tatsächlich übertragen werden. Dieser Level kann zur Laufzeit geändert werden.
8	Die Logger-Komponente benötigt zwei Software-Interfaces: **Logger**: Message erzeugen und eintragen. Eine Applikation kann via Methodenaufruf Messages (Textstrings) loggen. **LoggerSetup**: Dient zur Konfiguration des Message Loggers, z.B. setLevel(…) Weitere Interfaces sind wo sinnvoll individuell zu definieren.
9	**Statische Konfigurationsdaten** der Komponenten (z.B.: Informationen bezüglich Erreichbarkeit des Servers) sind zu definieren und müssen **ohne Programmierung anpassbar** sein.

2.4. Realisierung

Dieses Projekt muss vollständig in Java mit NetBeans[1] entwickelt werden. Es muss in den vorgegebenen Modulen in Subversion verwaltet und auf dem Buildserver laufend integriert werden.

Es geht darum, praktische Erfahrung bei Projektmanagement, Spezifikation, Design, Implementierung, Test, Dokumentation und Deployment eines komponentenbasierten Systems zu sammeln.

- Die Planung muss für ein iteratives Vorgehen mit mindestens zwei Iterationen aufgesetzt werden. Es ist davon auszugehen, dass ca. in der 8. Semesterwoche eine Zwischenabgabe (Prototyp) erfolgt und in der 9. Semesterwoche vom Dozierendenteam zusätzliche Vorgaben gemacht werden, die eine Überarbeitung und einen neuen Release des Message-Logger-Systems erfordern.
- Spezifikation und Design sind mittels der im Selbststudium (UML Kurzreferenz, Oestereich) erarbeiteten UML-Elemente zu beschreiben.
- Die Implementierung soll nur auf den Java Standard-Bibliotheken aufbauen.
- Mindestens für die Klassen der Logger-Komponente und des Logger-Servers sind JUnitTests Teil der geforderten Abgabe. Unit-Tests sind aber generell empfohlen.

[1] oder eine andere IDE, aber **nicht** mit BlueJ

Hochschule Luzern
Technik & Architektur

3. Vorgehen

3.1. Lernziele

- Sie können ein Softwareprojekt in kleinen Gruppen gemäss HT*Agil* in Iterationen durchführen. Dabei können Sie die Arbeitsteilung geeignet organisieren.
- Sie kennen den Wert klar definierter Anforderungen.
- Sie kennen die Vorteile von Komponenten.
- Sie können in einem Projekt mit beschränkten Ressourcen umgehen. Sie können sich im Spannungsfeld „Dauer-Umfang-Qualität-Kosten (= aufgewendete Zeit)“ bewegen.
- Sie können komponentenbasierte Architekturen und das Design von Komponenten mithilfe der UML angemessen und nachvollziehbar dokumentieren.
- Sie arbeiten mit zeitgemässen Werkzeugen und Methoden.
- Sie können eine Review organisieren und abwickeln.
- Sie kennen Verfahren, Werkzeuge und Bedeutung des Testens (insbesondere Unit- und Systemtest) und können diese Tests angemessen planen und Ergebnisse nachvollziehbar dokumentieren.
- Sie erstellen Unit-Tests und entwickeln exemplarisch nach dem Testfirst-Verfahren.
- Sie sammeln Erfahrung mit ‚Continuous Integration‘ (CI) durch die konsequente Nutzung von Versionskontrollsystem, automatisiertem Buildprozess und Buildserver.

3.2. Organisation der Gruppen

- Es sind **4er Teams** zu bilden. Sie machen die Zusammensetzung selbst und teilen diese dem Dozierenden mit.

3.3. Wichtige Punkte

3.3.1. Interface Komitee

Die Gruppen bestimmen beim Projektstart je eine/n Delegierte/n in die Interface-Komitees.
Es werden drei unabhängige Komitees gebildet, die je ein Set von Interfaces für die Logger-Komponente definieren und dokumentieren. Das Interface ist so zu gestalten, dass die entstehenden Logger-Komponenten **interoperabel** sind, d.h. **alle** Gruppen müssen dieses Interface verwenden.
Die Vorschläge werden dem Plenum präsentiert und es wird gemeinsam in Absprache mit dem Dozierendenteam entschieden, welches Set von Interfaces dann von allen Teams realisiert wird.
Die Komitees konstituieren sich selbst, d.h. sie bestimmen den Vorsitz des Interface Komitees sowie die Verantwortung für die Organisation, Zeitplanung, Dokumentation und Änderungsdokumentation (sofern notwendig).

3.3.2. Meilenstein-Review

Die Teams reviewen sich gegenseitig im Hinblick auf die Zwischenabgabe von SW9. Die Partner-Gruppe wird durch die Dozierenden bestimmt.
Die Partner-Teams überprüfen, ob die Artefakte (Dokumentation, Code, Tests etc.) den Anforderungen für die Zwischenabgabe fachlich genügen und vollständig sind. Die Teams erstellen einen Review-Bericht, dieser Bericht wird der Partner-Gruppe ausgehändigt und ist Teil von deren Zwischenabgabe.
Für die Organisation des Reviews sind die Gruppen selber verantwortlich. Jede Gruppe achtet darauf, dass für die eigene Arbeit rechtzeitig, d.h. spätestens SW8 ein Review-Bericht vorliegt.

Hochschule Luzern
Technik & Architektur

3.3.3. Präsentationen

Die eigene Lösung wird bei der Schlussabgabe in einer Präsentation vorgestellt.

Zwischenabgabe (SW 8)

Anwesenheitspflicht für alle eingeschriebenen Studierenden

- Demo des Message-Loggersystems, Release 1
- Vollständige Dokumentation (.pdf), Release 1
- Meilenstein Review durchgeführt und protokolliert
- Lauffähige Programme welche:
  - in den vorgegebenen Modulen in Subversion verwaltet werden
  - auf dem Buildserver laufend integriert wurden
  - automatisierte Tests ohne Fehler durchlaufen
  - eine begründete minimale Codeabdeckung erfüllen

Schlussabgabe (SW 13)

Anwesenheitspflicht für alle eingeschriebenen Studierenden

- Demo der Software mit eigener Loggerkomponente
- Demo der Integration einer fremden Loggerkomponente
- Vollständige Dokumentation (.pdf und ausgedruckt), Release 1 und 2
- Lauffähige Programme (analog zur Zwischenabgabe!)

Die Schwerpunkte der Präsentation und der genaue Zeitpunkt werden in Absprache mit den Dozierenden festgelegt.

3.4. Durchführung der Arbeit

Es stehen für die Arbeit ca. 1/3 der Unterrichtszeit sowie wesentliche Teile der Selbststudienzeit im Modul SWK zur Verfügung.

4. Beurteilung der Arbeit

Die Durchführung und die Abgabe der Arbeit ist Testatbedingung im Modul SWK. In Ausnahmefällen (ungenügende Mitarbeit in der Gruppe, kein ersichtlicher individueller Beitrag, fehlende Präsenz bei Zwischen- bzw. Schlussabgabe) kann das Testat einem einzelnen Gruppenmitglied verweigert werden.

Hochschule Luzern
Technik & Architektur

5. Organisation

5.1. Vorgehen und Termine

Projektlaufzeit: SW 1 bis SW 14

Die folgenden **Termine sind einzuhalten, die geforderten Arbeitsergebnisse sind Teil der Testatbedingung** (vollständige Teampräsenz für die Besprechung mit dem Dozenten, Termineinhaltung). Es ist den Teams überlassen zusätzliche und „schärfere" Termine zu planen. **Hinweis:** Alle Grundlagen für die Besprechungen mit den Dozierenden müssen *schriftlich* vorliegen.

	Inhalte der Besprechung
SW3	Organisation der Gruppe ist definiert; erste Risikoliste, sowie ein Rahmenplan mit Meilensteinen und eine erste grobe Aufwandschätzung liegen vor. Elaboration mit Prototyp zum Verständnis der RMI Schnittstelle ist geplant. Testplan: Testphilosophie und drei wesentliche Testaspekte. *Kurzbesprechung mit dem Dozenten, Protokoll durch Gruppe*
	Von den Interface-Teams liegen erste dokumentierte Versionen des Interfaces vor. => Do, 07. März 2013. 10:30 ILIAS Briefkasten *Präsentation und Besprechung im Plenum, Freigabe mit dem Dozenten.*
SW5	Freigegebene Version des Interfaces liegt vor. Elaboration ist abgeschlossen. Planung erste Iteration liegt vor: detaillierte Aufwandschätzungen, Arbeitsteilung in Gruppe, Organisation des Controlling etc. festgelegt. Peer Review sind organisiert (personell und zeitlich). Dokumentationsplan. Liste der Konfigurations-Items. Spezifikation der drei Elemente für das Systemtesting einschliesslich der Definition des Vorgehens liegt vor. *Kurzbesprechung mit dem Dozenten, Protokoll durch Gruppe*
SW5	Entwicklung am Release 1 aufgenommen. Code wird in Subversion verwaltet und laufend integriert. (Geforderte) Unittests laufen erfolgreich. *Alle Projekte sind auf dem Buildserver fehlerfrei buildbar, es liegen Testfälle vor, Kontrolle durch Dozent*
SW9	Demonstration / Präsentation (Zwischenabgabe) Softwareanforderungen erstellt. Architektur ist festgelegt und exemplarisch dokumentiert. Prototyp ist lauffähig und kann demonstriert werden. Abgabe der Dokumentation & Projektcontrolling (elektronisch) => Do, 18. Apr. 2013. 12:00 ILIAS Briefkasten, *Kurzbesprechung mit dem Dozenten, Protokoll durch Gruppe*
SW14	Aktualisierte Planung und nachgeführte Softwarespezifikation liegen vor. Alle Komponenten sind lauffähig und können demonstriert werden. Die Interoperabilität der Logger-Komponente kann demonstriert werden. Demonstration / Präsentation (Schlussabgabe) Abgabe der Dokumentation (elektronisch + Papier). => Fr, 24. Mai 2013. 12:00 ILIAS Briefkasten, *Kurzbesprechung mit dem Dozenten, Protokoll durch Gruppe*

Projektauftrag: Message-Logger | Neue Anforderungen

Hochschule Luzern
Technik & Architektur

Modul: TA SWK FS2013

Projektauftrag:Message-Logger
Neue Anforderungen

V1.0, April 2013; M. Jud, R. Gisler, P. Sollberger

1. Neue Anforderungen

Ihre Geschäftsleitung hat beschlossen, der *CLoSIG* (**C**ommon **Lo**g**S**erver **I**nterface **G**roup) beizutreten. Die *CLoSIG* empfiehlt für die Kommunikation ***zwischen der Logger-Komponente und dem Logger-Server*** CORBA zu verwenden.

Die Geschäftsleitung gibt der Entwicklungsabteilung die folgenden Aufträge

- den Logger-Server so anzupassen, dass die Schnittstelle für die Logger-Komponente entsprechend den Empfehlungen der *CLoSIG* ***zusätzlich*** auch CORBA unterstützt.
- eine neue CORBA-fähige Logger-Komponente zu entwickeln.

Dabei legt die Geschäftsleitung besonderen Wert darauf, dass Anwendungen, welche bisher Ihren Logger verwendet haben ***ohne Quellcodeanpassungen*** in den bestehenden Anwendungen auf die neue, CORBA-fähige Logger-Komponente umsteigen können.

2. Vorgehen

1. Das Interface Team ***definiert*** das IDL Interface zwischen der neuen Logger-Komponente und erweitertem Logger-Server.

2. Jedes Team ***plant*** (Projektplan) und ***dokumentiert*** (SysSpec) die nächste Iteration im SWK Logger-Projekt basierend
   - auf dem Stand der Zwischenabgabe einerseits
   - und auf diesen neuen Anforderungen

3. Jedes Team ***implementiert in seinem Logger-Server*** zusätzlich den CORBA Server, (die bereits bestehende, proprietäre Socket-basierende Schnittstelle/Kommunikation muss weiter funktionieren).

4. Jedes Team ***erstellt eine zweite, unabhängige Logger-Komponente***, welche
   - die gegenüber der DemoApplikation bestehenden Logger und LoggerConfig-Schnittstellen implementiert und einhält (wie die bereits für die Zwischenabgabe realisierte Komponente).
   - Meldungen an den Logger-Server jedoch über das CORBA Interface übermittelt (statt über das proprietäre Interface der für die Zwischenabgabe realisierten Komponente).

5. Zeigen Sie mit ***Tests***, dass
   - Ihre DemoApplikation ohne Codeanpassungen d.h. so, wie für die Zwischenabgabe realisiert, alternativ mit einer proprietären Komponente als auch mit einer neuen, gegenüber dem Server CORBA-fähigen Komponente loggen kann (im Austausch).
   - Ihr Logger-Server gleichzeitig über beide Schnittstellen Log-Meldungen entgegennehmen und ablegen kann.

Hochschule Luzern
Technik & Architektur

3. Hinweise und Empfehlungen

- Falls Ihrer Software die Anforderungen für die Zwischenabgabe noch nicht erfüllt hatte, stellen Sie zuerst unbedingt sicher, dass Sie einen stabilen, funktionsfähigen Release auf diesem Stand haben.
- Verwenden Sie Entwurfsmuster. Eventuell können Sie damit viele Teile wieder verwenden.
- Führen Sie Ihre Dokumentation so nach, dass sowohl Ihre proprietäre Logger-Komponente (Zwischenabgabe) als auch die neue CORBA-Logger-Komponente angemessen dokumentiert sind.
- Das Interface-Team kann die notwendige Code-Generierung für Java übernehmen und das IDL komplett mit Stub- und Skeletonklassen in einem Release liefern.
- Die Wahl, mit welcher Komponente Ihre Testapplikation loggt, soll beim Start festgelegt werden können (Startparameter, Property-Datei, Classpath o.ä. – auf jeden Fall OHNE Anpassung der Quellen und Neukompilation).