Team 2

Michael Kressibucher Sandro Wicki Trong Viet Dao Oliver Summermatter

Logger

System-Spezifikation

1. Systemübersicht 2
2. Architektur und Designentscheide 3
2.1. Modell(e) und Sichten 3
2.2. Daten (Mengengerüst & Strukturen) 8
2.3. Entwurfsentscheide 9
3. Schnittstellen 12
3.1. Externe Schnittstellen 12
3.2. wichtige interne Schnittstellen 12
3.3. Benutzerschnittstelle(n) 12
4. Environment-Anforderungen 13

Versionen:

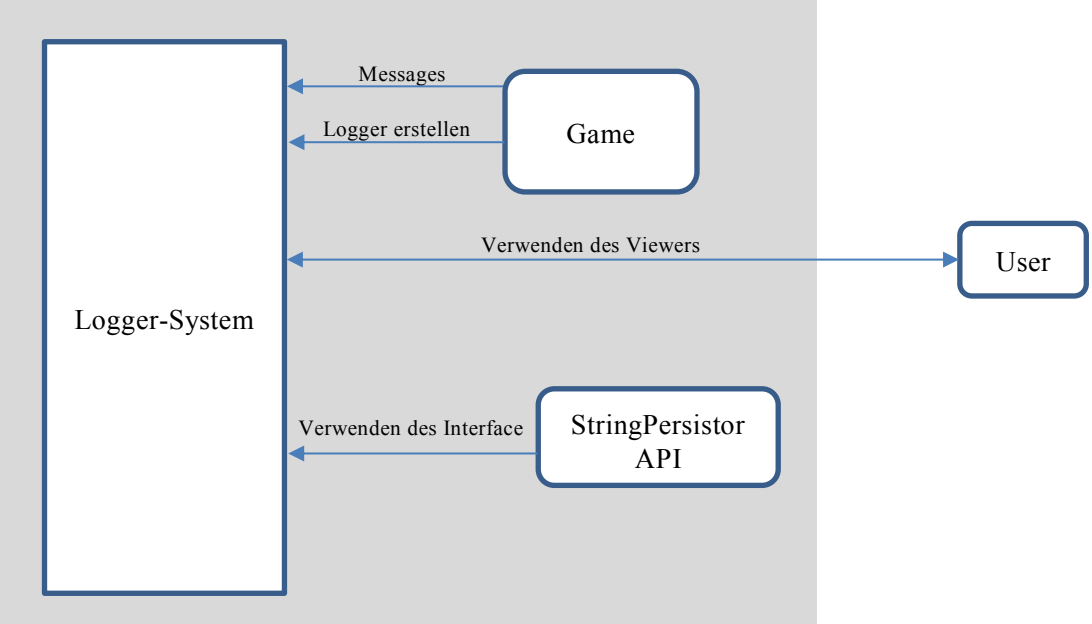
Rev.	Datum	Autor	Bemerkungen	Status
0.1			1. Entwurf	fertig
1.0	5.11.2017	Alle	Definitive Version 1.0	fertig

1. Systemübersicht

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

1.1. Systemgrenzen

Die Systemgrenzen des Projektes werden in der folgenden Abbildung dargestellt. Die Game-Komponente erstellt den Logger und schickt diesem die Messages zu welche dieser zu loggen hat. Der User hat die Möglichkeit über den Viewer mit dem Logger-System zu interagieren. StringPersistor API stellt das Interface zur Verfügung für den StringPersistor.

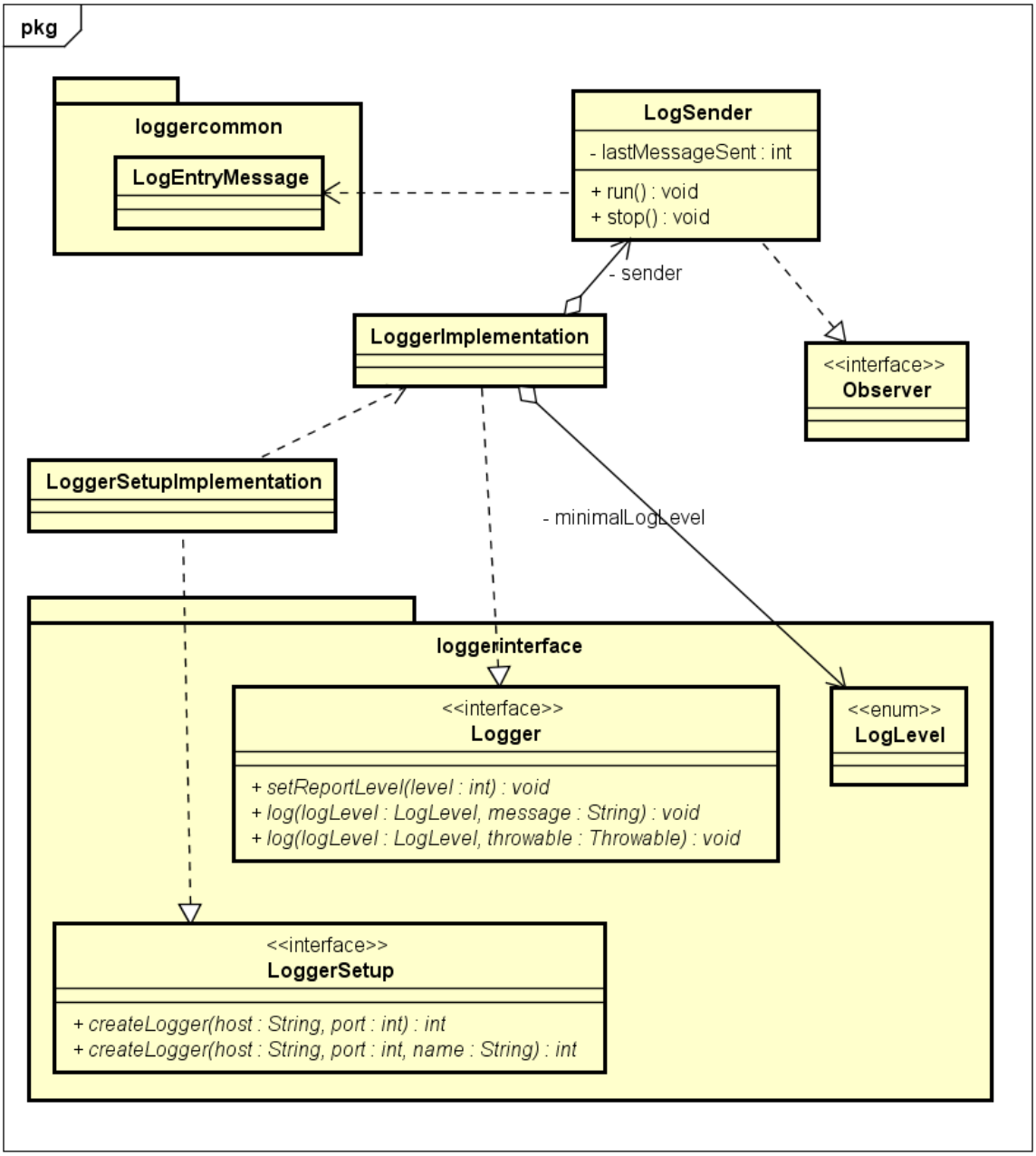


2. Architektur und Designentscheide

2.1. Modell(e) und Sichten

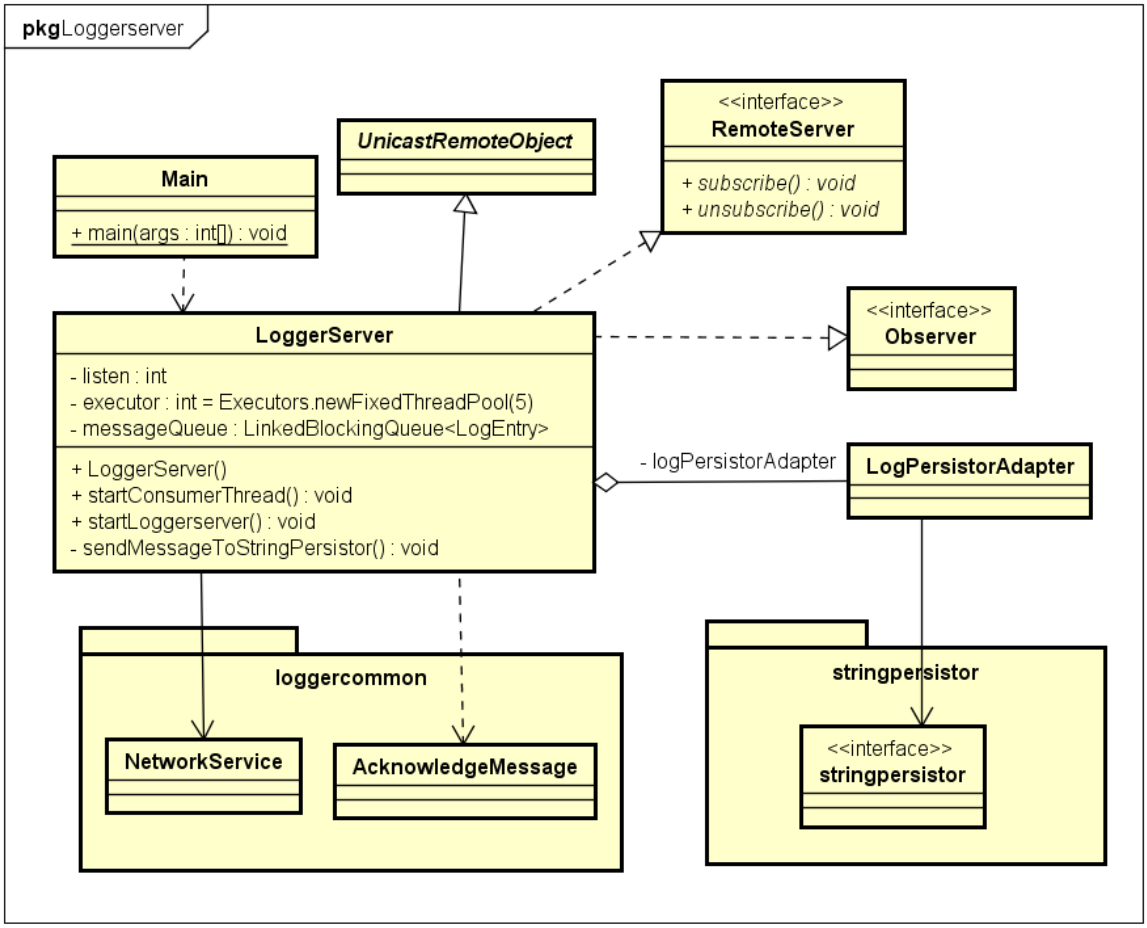
2.1.1. Komponente Loggercomponent

Die Klasse LoggerImplementation erbt von der Klasse Application, welche den Buffer für AbstractMessages enthält. Beim Aufruf der Methode werden die übergebenen Informationen in den Buffer gespeichert. Die Klasse LogSender wird in einem neuen Thread gestartet und sendet die im Buffer gespeicherten Messages an den Server. Nachfolgend das Klassendiagramm der Komponente Logger.

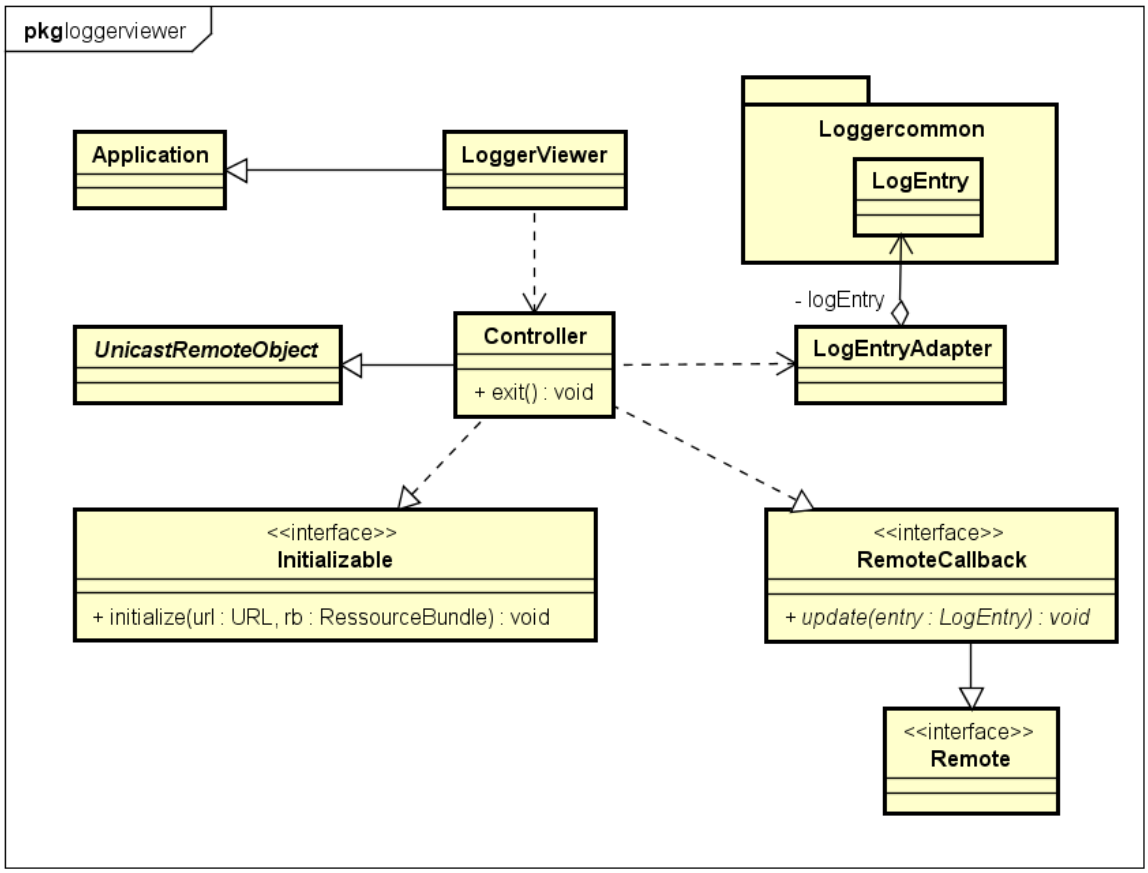


2.1.2. Komponente Loggerserver

Die Klasse LoggerServer wird über die Methode startLoggerserver gestartet. Der LoggerServer akzeptiert TCP-Verbindung von Clients und startet einen NetworkService in einem neuen Thread des Thread-Pools. Nachfolgend das Klassendiagramm der Komponente Loggerserver.

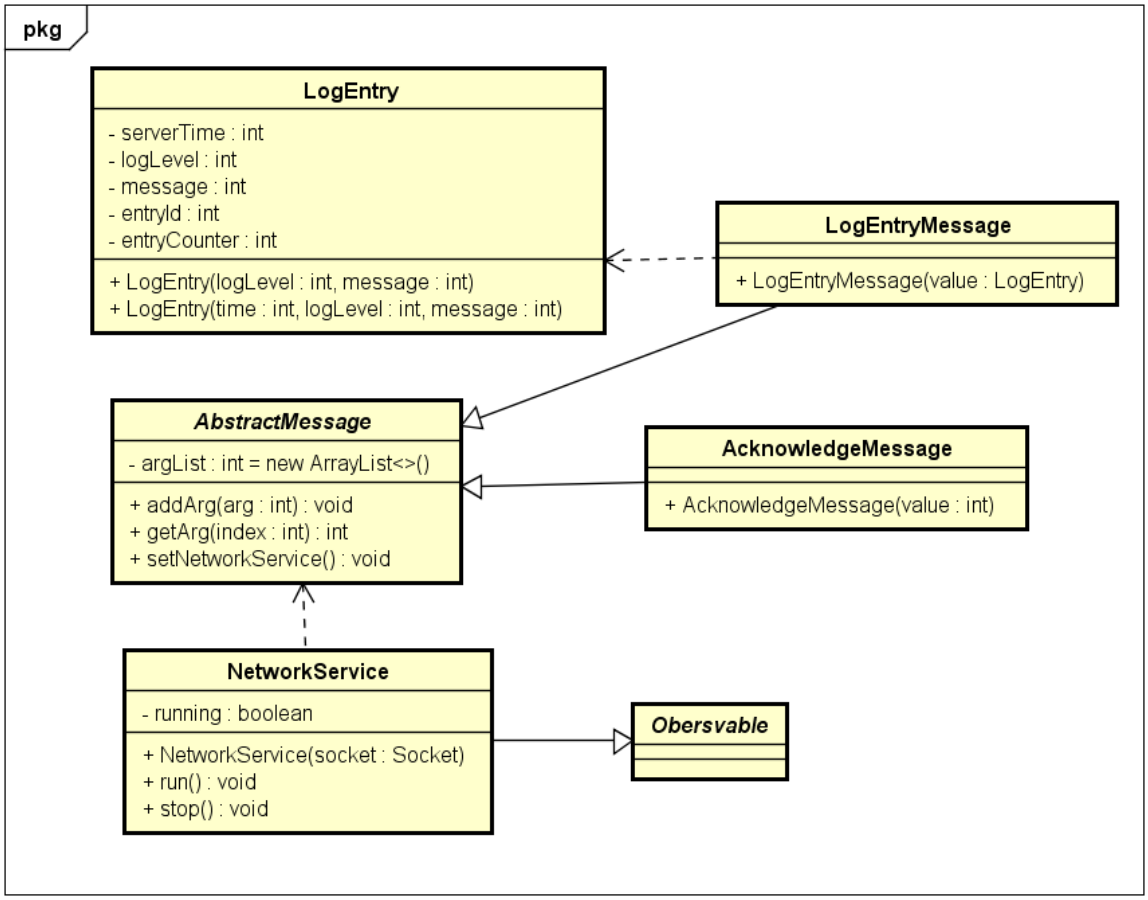


2.1.3. Package Loggerviewer



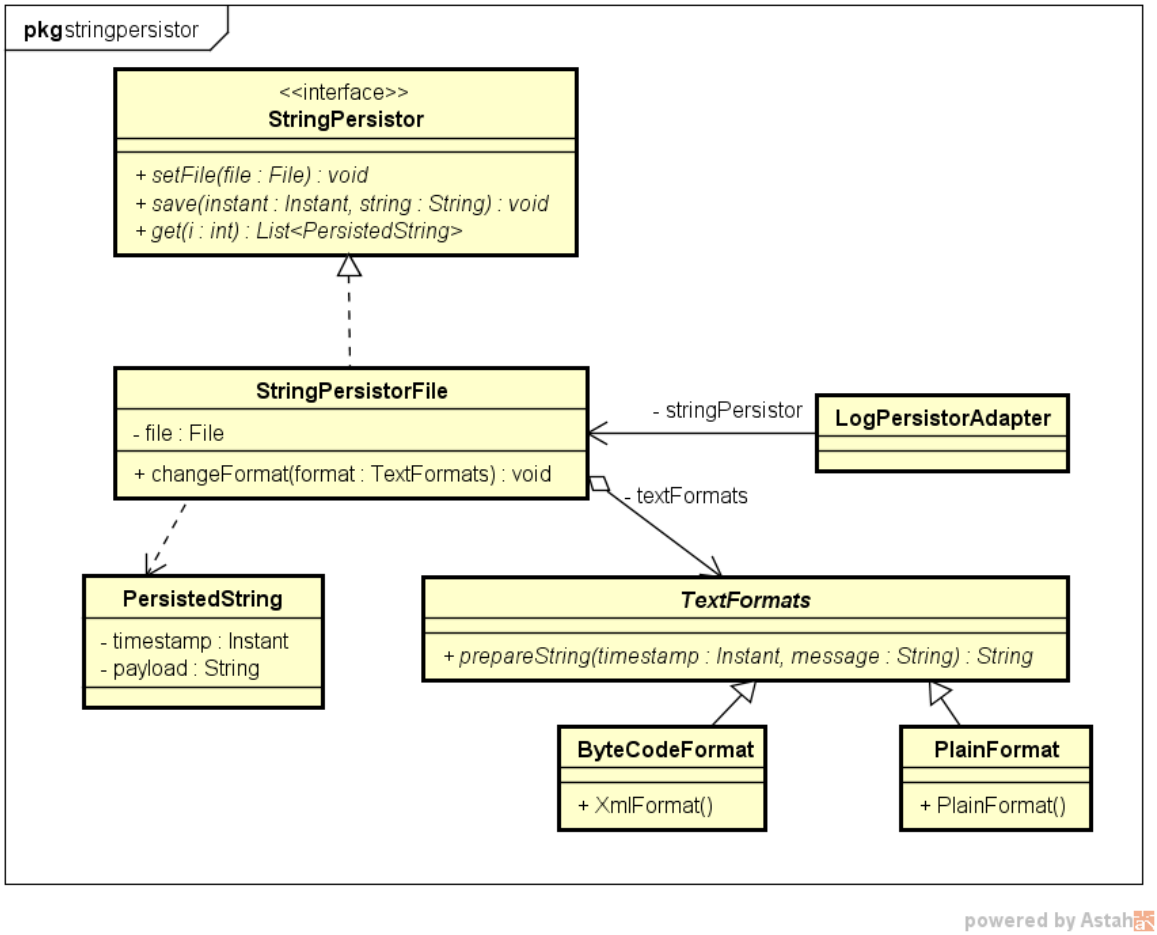
2.1.4. Package Loggercommon

Dieses Package enthält die Klassen und Interfaces, welche die Komponente Logger und Loggerserver gemeinsam benötigen. Nachfolgend das Klassendiagramm des Package Loggercommon.



2.2. Komponente Stringpersistor

In der Komponente wurden das Strategy- und das Adapter-Pattern verwendet, wie im Kapitel Entwurfsentscheide beschrieben. Nachfolgend das Klassendiagramm der Komponente Stringpersistor.



2.3. Daten (Mengengerüst & Strukturen)

Die Konfigurationen für den Logger und die persistierten Logeinträge werden momentan als Plain Text von einer Textdatei gelesen und gespeichert.

Die Konfigurationsdatei ist auf dem lokalen System des Loggers vorhanden. In dieser Datei sind die Daten für die Verbindung mit dem Logserver gespeichert. Folgende Daten sind enthalten und werden vom Logger nur gelesen:

- **IP-Adresse:** Gibt die IP-Adresse des Logservers an.
- **Port:** Beschreibt den Port über den der Logger den Server erreichen kann.
- **Classname:** Die eingesetzte Klasse um ein LoggerObjekt zu erstellen.

ip-address = 127.0.0.1
port = 4242
classname = ch.hslu.vsk.LoggerSetupImplementation

Der Logger Server erhält vom Logger ein serialisiertes LogEntry-Objekt, dass er dann deserialisiert und im String Persistor übergibt. Das LogEntry-Objekt enthält folgende Daten:

- **ServerTime:** Die ServerZeit wird beim Erhalten noch nachgetragen.
- **LocalTime:** Beschreibt die lokale Zeit, an der der Logeintrag beim Logger erstellt wurde.
- **LogLevel:** Beschreibt die Wichtigkeit der Nachricht.
- **Message:** Die erhaltene Nachricht des Loggers, die persistiert werden soll.
- **LoggerName:** Ein Name um den Logger zu identifizieren.

Der String Persistor erhält den aktuellen Zeitstempel und den Logeintrag als String. Dieser wird dann im folgenden Format in die Logdatei geschrieben:

ServerTime@LocalTime; LoggerName; LogLevel, Message
Beispiel
2017-11-05T15:14:18.344Z@2017-11-05T15:14:17.173Z; loggerg02; INFO; Zoom changed to 7

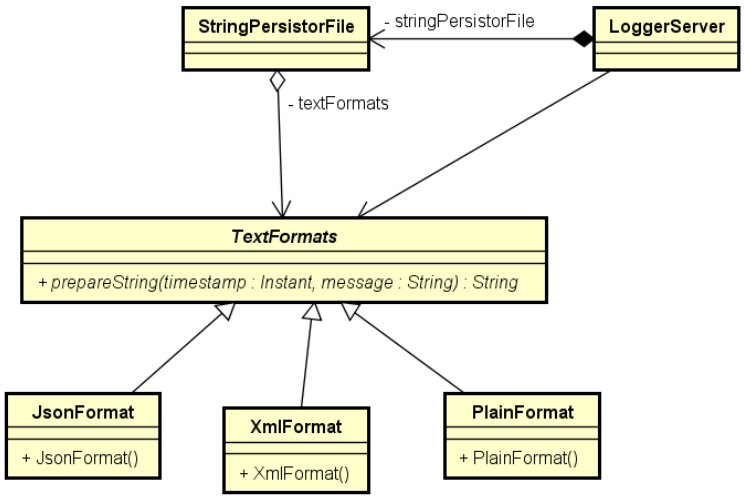
2.4. Entwurfsentscheide

In diesem Kaptiel werden die wichtigsten Entwurfsentscheiden, welche im Projekt getroffen werden dokumentiert. Die folgenden Entwurfsmuster wurden im Projekt eingesetzt:

- Strategie
- Message Passing
- Adapter

2.4.1. Strategie

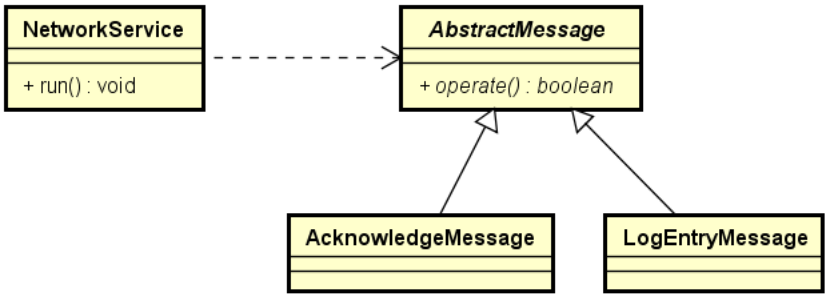
Das Strategie Pattern wird in diesem Projekt eingesetzt, um die Log-Einträge in verschiedenen Formaten abspeichern zu können.



Wir haben uns dazu entschieden das Strategie-Pattern einzusetzen, da so das Speicherformat einfach angepasst werden kann. Ebenfalls können leicht neue Speicherformate hinzugefügt werden.

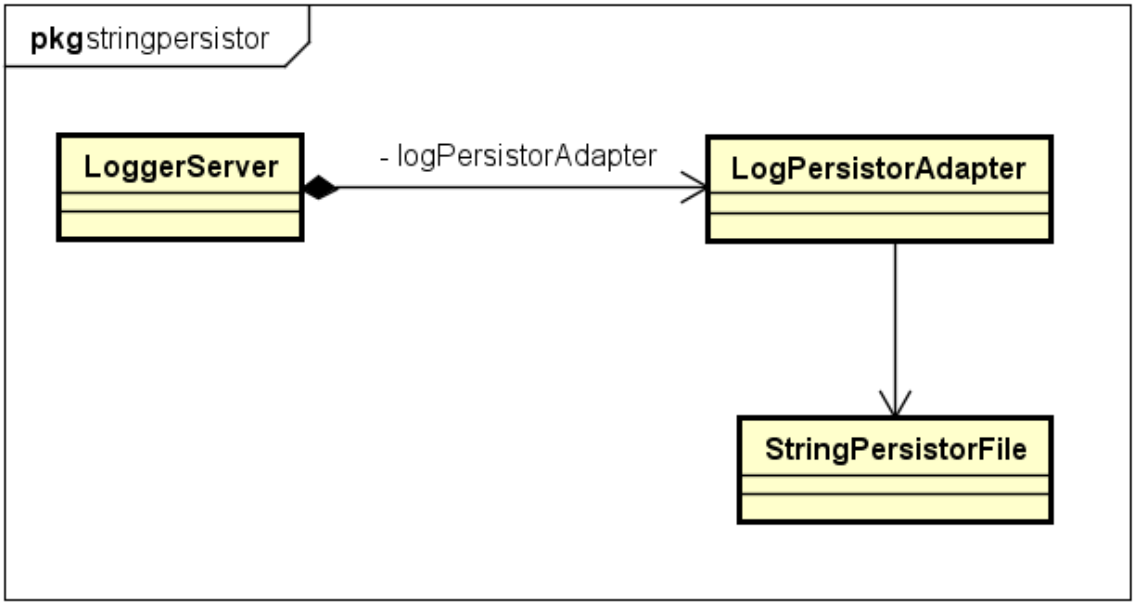
2.4.2. Message Passing

Wir haben uns für das Message Passing Pattern entschieden, da mit diesem Pattern einfach neue Message-Typen hinzugefügt werden können. Somit kann die Applikation an neue Anforderungen angepasst werden. Jedoch haben wir nicht eine der Vorgestellten Codeskizzen vollständig übernommen, sondern diese mit der Objekt-Serialisierung implementiert.



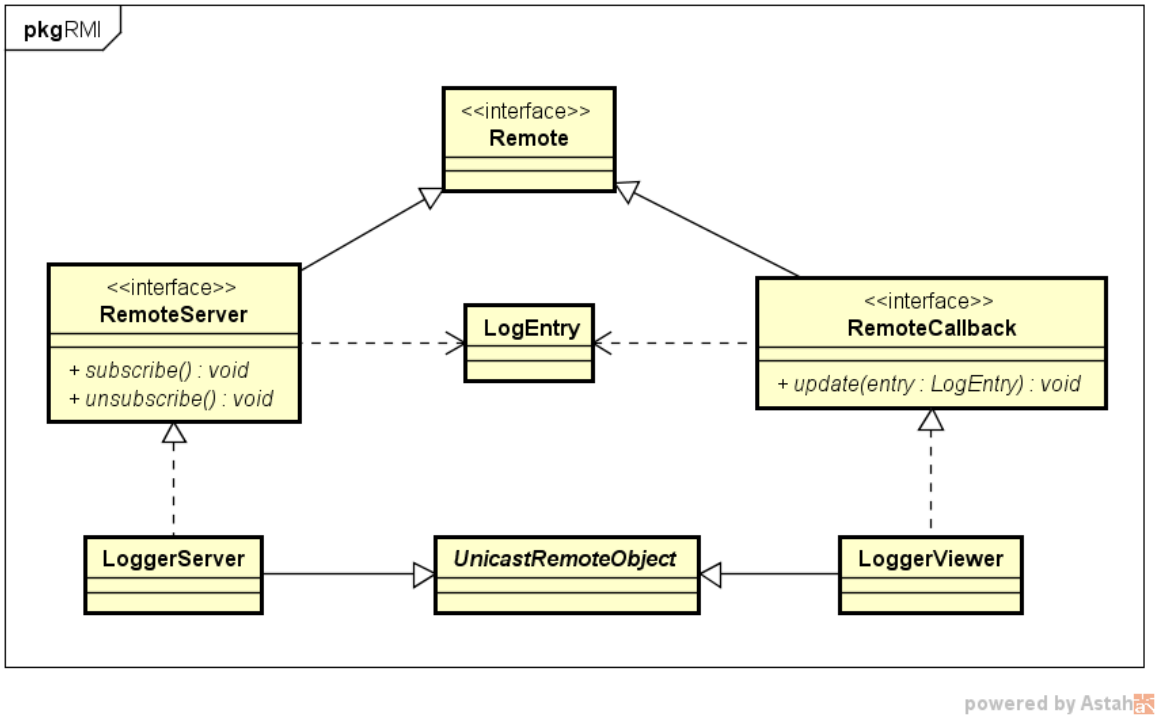
2.4.3. Adapter

Das Adapter-Pattern wird in diesem Projekt eingesetzt, um die StringPersistor-Schnittstelle besser im Projekt integrieren zu können. Der LogPersistorAdapter-Klasse können in der save-Methode ein LogEntry-Objekt anstatt eines Strings übergeben werden.



2.4.4. RMI

RMI (Remote method invoke) wird verwendet, um eine Push Synchronisierung zwischen Server und LoggerViewer zu erstellen. Durch die Verwendung von RMI kann sich der Viewer beim Server registrieren, um informiert zu werden, wenn ein neues Log-Event eintrifft.



2.4.5. Uhren Synchronisation

Logische Uhren können genutzt werden um die Log-Einträge kausal korrekt zu speichern. Konkret könnte dies bei der Übertragung an den Server erfolgen.
Im Projekt werden keine logischen Uhren eingesetzt da im Loggersystem die Prozesse oft ändern und folglich der Aufwand zu gross wäre diese zu synchronisieren.

Alle Nachrichten sind mit Sicherheit in der korrekten Reihenfolge gespeichert, somit können keine Fehlinterpretationen gemacht werden.

Die Implementation des Lamport-Zeitstempel würde in diesem Fall mehr Sinn machen, da die Nachrichten nur in eine Richtung versendet und demensprechend die Zeiten in eine Richtung synchronisiert werden.

Im Projekt synchronisieren der Client und der Server ihre Uhren per NTP mit einer externen Atomuhr.

3. Schnittstellen

3.1. Externe Schnittstellen

In diesem Projekt wurden folgende externe Schnittstellen verwendet:

- LoggerInterface Version: 1.0.0
- StringPersistor Version: 4.0.0

Die beiden Schnittstellen konnten vom Binary-Repo der HSLU bezogen werden.

3.2. wichtige interne Schnittstellen

3.2.1. Schnittstelle TCP/IP

3.2.1.1. Steckbrief

Diese Schnittstelle definiert den Vorgang um Ereignisse vom Logger auf den Logger-Server zu übertragen. Sie wurde vom Team 2 erstellt.

3.2.1.2. Interaktionen

Die Verbindung wird von der Java-Socket-Klasse aufgebaut. Der Server nimmt diese eingehenden Verbindungen entgegen und startet einen neuen NetworkService. Der Server kann somit Nachrichten von mehreren Instanzen empfangen.

Die Daten werden in einer abgeleiteten Message, der abstrakten AbstractMessage-Klasse, welche serialisiert wurde, übertragen. Die AbstractMessage-Klasse hat die Attribute MessageID und eine List mit Argumenten.
Zur Kommunikation über die TCP/IP-Schnittstelle werden die Nachrichtentypen AcknowledgeMessage und LogEntryMessage verwendet. Die LogEntryMessage wird gesendet, wenn ein Log-Event an den Server geschickt werden soll. Die AcknowledgeMessage dient dazu eine erhaltene LogEntryMessage zu bestätigen.

Die folgenden Fehlersituationen sind bei dieser Schnittstelle möglich:

- Die Verbindung zwischen den Komponenten der Schnittstelle wurde getrennt. Durch den Verbindungsunterbruch können keine Nachrichten mehr gesendet oder empfangen werden. Um den Fehler zu beheben wird das Senden von Nachrichten unterbrochen und versucht eine neue Verbindung aufzubauen. Wenn eine neue Verbindung aufgebaut wurde, werden alle unbestätigten erneut gesendet.

3.2.1.3. Einstellungen

Der Schnittstelle können neue Message-Typen hinzugefügt werden.

3.2.1.4. Qualitätsmerkmale

Die Schnittstelle muss in der Lage sein bis zu 50 Messages/s zu übertragen.
Bei einem Netzwerkunterbruch gehen keine Nachrichten verloren.

3.2.1.5. Entwurfsentscheidungen

Die Schnittstelle wurde erstellt, um die Kommunikation zwischen dem Logger und dem Logger-Server zu definieren.

3.2.1.6. Beispielverwendung

Pseudo-Code:

```
        Schleife
                Nachricht senden
                Überprüfen, ob neue Nachrichten vorhanden sind
neuer Thread
        Schleife
                Empfangen einer Bestätigungs-Nachricht
```

3.3. Benutzerschnittstellen

Innerhalb dieses Projektes gibt es keine spezifischen Benutzerschnittstellen.

4. Environment-Anforderungen

Diese Logger Software benötigt kein spezielles Betriebssystem. Sie benötigt jedoch ein Gerät, welches die folgenden Voraussetzungen erfüllt:

- Java 8 RE installiert
- Netzwerkverbindung möglich
- Port 4242 ist nicht besetzt