

Systemspezifikation

Version 1.0.0

Gruppe 5 (Patrick Bucher, Pascal Kiser, Fabian Meyer, Sascha Sägesser)

07.04.2018

Inhaltsverzeichnis

| | |
|--|----------|
| 1 Systemübersicht | 1 |
| 1.1 Implementierungsspezifische Komponentenarchitektur | 2 |
| 1.2 Kontextdiagramm | 2 |
| 2 Architektur und Designentscheide | 2 |
| 2.1 Modelle und Sichten | 4 |
| 2.2 Datenstrukturen | 4 |
| 3 Schnittstellen | 5 |
| 3.1 Externe Schnittstellen | 5 |
| 3.2 Interne Schnittstellen | 5 |
| 4 Implementierung | 6 |
| 4.1 Zwischenabgabe | 6 |
| 5 Environment-Anforderungen | 6 |

1 Systemübersicht

Die Systemarchitektur ist grösstenteils durch den Projektauftrag festgelegt. Dieses ist in der Abbildung **Komponentendiagramm Projektauftrag** ersichtlich.

Die Anwendung besteht aus zwei Bereichen: Dem Client und dem Server. Das *Game* kommuniziert mittels *Logger*- und *LoggerSetup*-Schnittstelle mit der *LoggerComponent*. Diese schlägt die Brücke zum Server-Bereich über ein TCP/IP-Protokoll, worüber sie mit dem *LoggerServer* kommuniziert. (Dieses Protokoll ist implementierungsspezifisch und bei der Gruppe 5 im Dokument *TCP-Schnittstelle* dokumentiert.) Der *LoggerServer* macht Gebrauch von einer Komponente namens *StringPersistorFile*, mit welcher er über die *StringPersistor*-Schnittstelle kommuniziert.

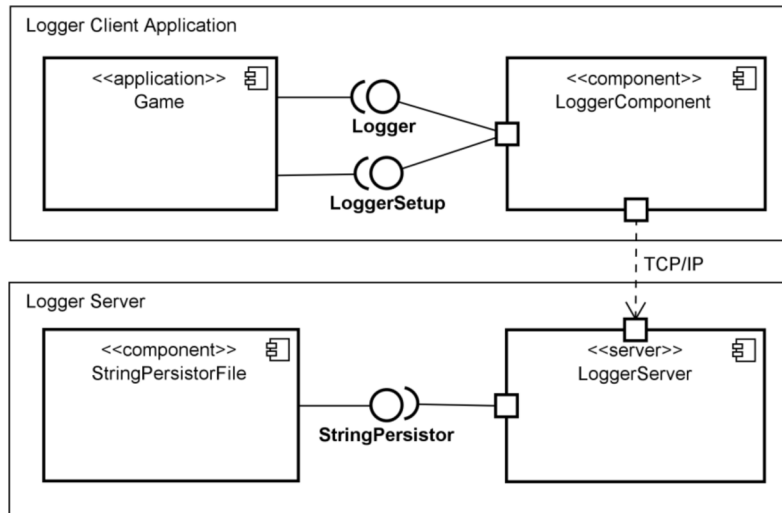


Abbildung 1: Komponentendiagramm Projektauftrag

1.1 Implementierungsspezifische Komponentenarchitektur

Die Abbildung **Komponentendiagramm** gibt das Komponentendiagramm aus dem Projektauftrag leicht verändert wieder, indem es dem Umstand Rechnung trägt, dass der *LoggerServer* und die Komponente *StringPersistorFile* nicht direkt über die *StringPersistor*-Schnittstelle, sondern per *LogPersistor*-Schnittstelle über den *StringPersistorAdapter* und die *StringPersistor*-Schnittstelle miteinander kommunizieren.

1.2 Kontextdiagramm

Die Abbildung **Kontextdiagramm** bietet einen abstrakten Überblick über das System und dessen Kontext. Zum System gehört die gesamte Applikation.

Die Schnittstelle zwischen dem System und dem Benutzer stellt z.B. ein Computer dar, auf dem der Benutzer dann das *Game of Life* spielen und gleichzeitig loggen kann. Zusätzlich gehört die Aufgabenstellung, sprich der *LoggerProjektauftrag*, zum Kontext, da das System auf Basis dessen entwickelt wird.

2 Architektur und Designentscheide

Die Architektur wurde weitgehendst vom Auftraggeber vorgegeben. Diese können grösstenteils im Dokument *Projektauftrag* nachgelesen werden bzw. sind bereits eingangs in diesem Dokument aufgeführt.

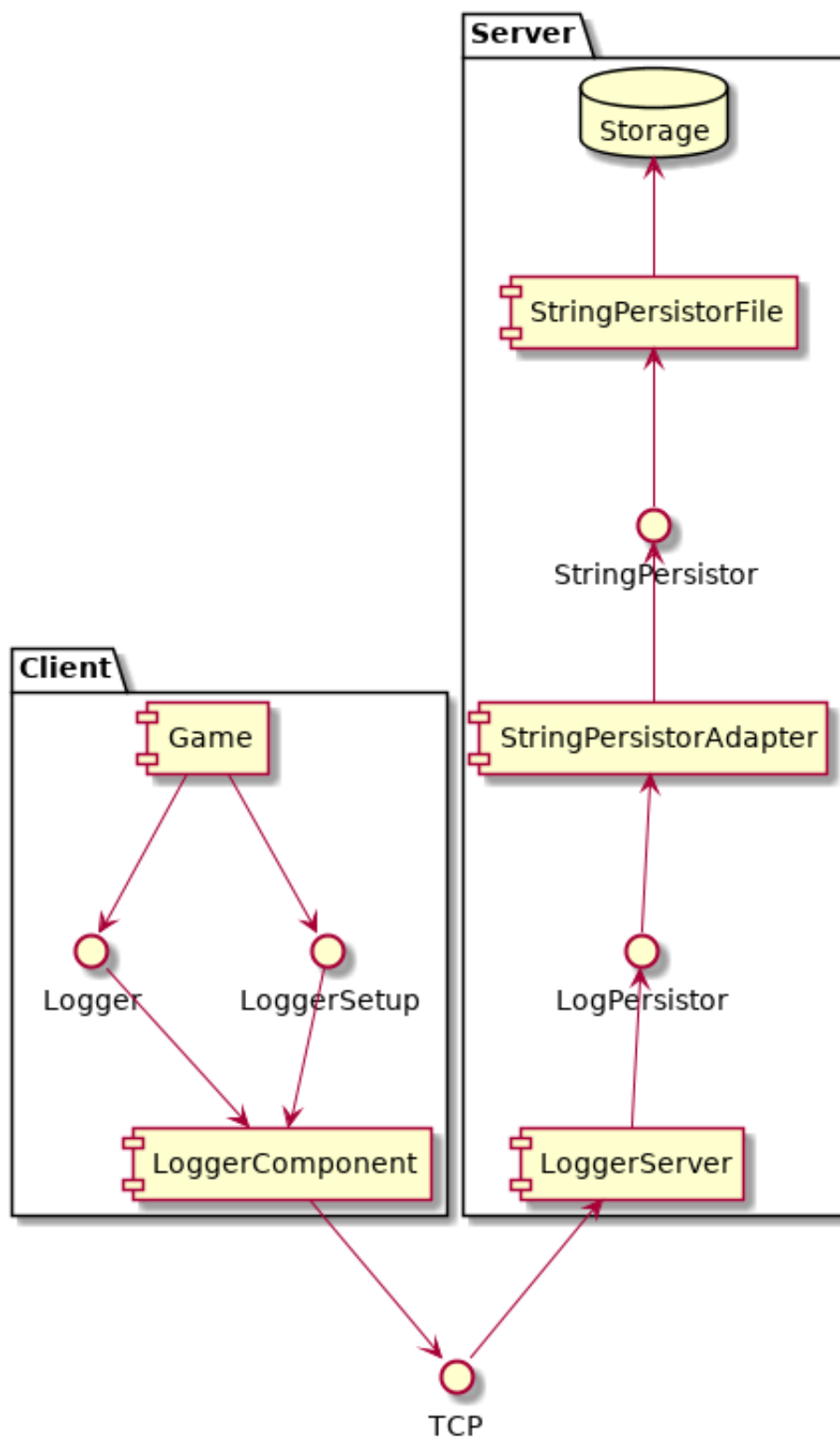


Abbildung 2: Komponentendiagramm

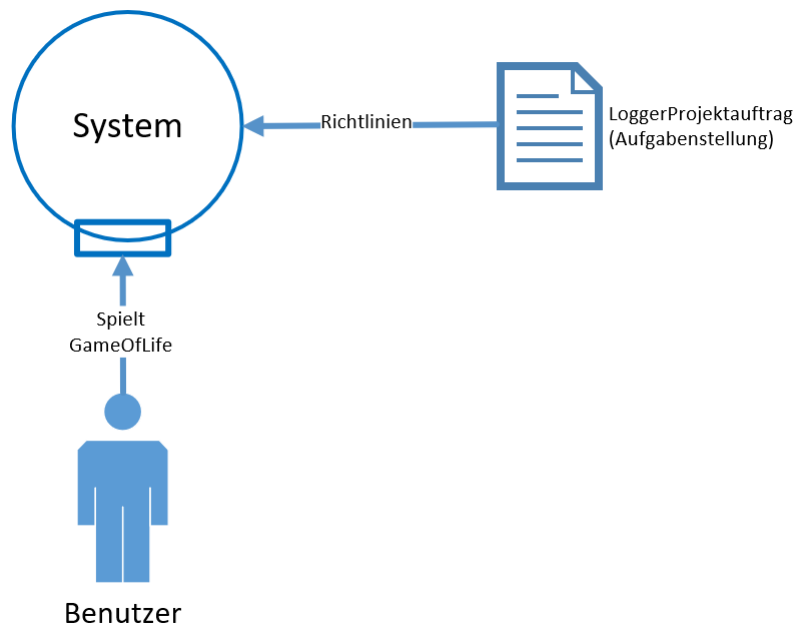


Abbildung 3: Kontextdiagramm

2.1 Modelle und Sichten

Beim Formulieren der Scrum-Stories (siehe Dokument *Scrum*) wurde die Anwendung von drei Perspektiven aus betrachtet:

- Ein *Anwender* führt die die *Game of Life*-Applikation aus und will seinen Spielstand geloggt wissen.
- Ein *Administrator* führt Konfigurationsarbeiten aus und will Parameter wie Log-Level und Serverkoordinaten einstellen können.
- Ein *Programmierer* will die Software-Komponenten zur Verfügung haben, um damit die Anforderungen von *Anwender* und *Administrator* umsetzen zu können: z.B. eine Logger-Komponente mit entsprechenden Interface, damit der die Logger-Aufrufe in die Anwendung einführen kann.

Da alle Projektmitarbeiter und selbst die Auftraggeber die Anwendung aus allen der genannten Perspektiven betrachten und diese gar von innen her kennen, erübrigt sich eine tiefgehende Diskussion über Perspektiven.

2.2 Datenstrukturen

Für den Austausch von Log-Meldungen wurden zahlreiche Datenstrukturen definiert:

- *Message* ist eine serialisierbare Klasse im *logger-common*-Projekt und dient als Austauschcon-

tainer für Log-Meldungen zwischen `LoggerComponent` und `LoggerServer`. Sie besteht nur aus `String`-Attributen.

- Die Schnittstelle `LogMessage` und deren Implementierung `LogEntry` dienen als Austauschcontainer für Log-Meldungen zwischen `LoggerServer` und `StringPersistor` über den `StringPersistorAdapter`. Diese kann flexibel mit verschiedenen `LogMessage`-Implementierungen umgehen.
- Die Klasse `PersistedString` dient zum Abspeichern und späteren Auslesen von Log-Meldungen durch den `StringPersistor`.

Wie diese Datenstrukturen genau von den verschiedenen Komponenten verwendet werden, kann dem Dokument *Klassendiagramm* entnommen werden.

3 Schnittstellen

Im Rahmen des vorliegenden Projektes wurden mehrere Schnittstellen definiert. Diese wurden einerseits vom Auftraggeber festgelegt (`StringPersistor`) bzw. in den Interface-Komitees definiert (`Logger` und `LoggerSetup`), wobei der Auftraggeber den entsprechende Rahmen vorgegeben hat; andererseits in den jeweiligen Projektteams ausgearbeitet.

3.1 Externe Schnittstellen

Um einen Austausch der `Logger`-Komponente mit Implementierungen anderer Gruppen zu ermöglichen, wurden folgenden Schnittstellen definiert:

| Schnittstelle | Version |
|------------------------------|----------------|
| <code>Logger</code> | 1.0.0-SNAPSHOT |
| <code>LoggerSetup</code> | 1.0.0-SNAPSHOT |
| <code>StringPersistor</code> | 4.0.1 |

Die Schnittstellen `Logger`, `LoggerSetup` und `StringPersistor` werden hier nicht weiter beschrieben. Ihre Dokumentation finden sich auf ILIAS. Von Java 1.8 zur Verfügung gestellte Schnittstellen sind hier nicht aufgeführt.

3.2 Interne Schnittstellen

Neben den externen, von aussen vorgegebenen Schnittstellen enthielt der Projektauftrag auch Anforderungen, welche die Definition weiterer Schnittstellen erforderlich machten. Dies sind:

| Schnittstelle | Version |
|---------------------------|----------------|
| <code>LogPersistor</code> | 1.0.0-SNAPSHOT |

| Schnittstelle | Version |
|---------------|----------------|
| TCP-Protokoll | 1.0.0-SNAPSHOT |

Die Schnittstelle `LogPersistor` wurde selber definiert. Sie bildet die Grundlage für den `StringPersistorAdapter` und stellt dem `LoggerServer` eine für diesen geeigneten Schnittstelle zum `StringPersistor` dar. Hierzu wurde das Adapter-Pattern (GoF 139) implementiert. Auf das TCP-Protokoll wird im Dokument *TCP-Schnittstelle* näher eingegangen.

4 Implementierung

Auf die Implementierung der Anwendung soll an dieser Stelle nicht weiter eingegangen werden. Stattdessen sei hier auf die JavaDoc und auf das umfassende Klassendiagramm (Dokument *Klassendiagramm*) hingewiesen. Weitere Hinweise zur Implementierung finden sich in den Dokumenten *Testplan* und *TCP-Schnittstelle*.

4.1 Zwischenabgabe

Für die Zwischenabgabe ist weiter zu beachten, dass die gegenwärtige Implementierung *nicht* für die persistente Aufzeichnung von Logmeldungen ausgelegt ist. Die Logdateien werden derzeit im temporären Verzeichnis des jeweiligen Benutzers gehalten. Dies führt dazu, dass bei einem Systemneustart sämtliche Daten verloren gehen. Für die Schlussabgabe soll das System dahingehend erweitert werden, dass das Log-Verzeichnis auf dem Server konfigurierbar ist.

5 Environment-Anforderungen

Zum Ausführen der Anwendung wird client- wie serverseitig die [Java SE Runtime Environment 8](#) benötigt. Tests mit der Open-Source-Variante *OpenJDK 8* sind problemlos verlaufen.

Zum Kompilieren der Anwendung wird Maven und das HSLU-Nexus-Repository benötigt. Weiter wurde zum bequemen Packen und Ausführen von Client und Server jeweils ein `Makefile` geschrieben, das sich im Wurzelverzeichnis des Projekts `g05-game` (Client) bzw. `g05-logger` (Server) befindet¹.

Zum Ausführen der Anwendung auf zwei verschiedenen Rechnern (Client und Server) wird eine funktionierende Netzwerkverbindung zwischen diesen beiden verlangt. Die beiden Systeme sollten zum gleichen Netzwerk gehören, da eine wechselseitige TCP-Kommunikation über einen Port > 1024 oftmals von Firewalls (d.h. an den Netzwerkgrenzen) unterbunden wird.

¹Ironischerweise ist Maven angetreten um ant zu ersetzen, welches wiederum angetreten ist um make zu ersetzen. Die Lösung mit dem `Makefile` macht das Zusammensuchen und Ausführen (mit der entsprechenden `-classpath`-Option aber äusserst bequem, da make die Vorzüge leichtgewichtigen Dependency-Managements und der Shell kombiniert.