

**Technisches Konzept**

**Projektname: M-Wetter**

**Version:** 0.1

**Status:** in Arbeit

**Datum:** 07.07.2017

**Autor:** Sonja Wasenmiller und Tobias Gilnhammer

Inhaltsverzeichnis

[Inhaltsverzeichnis 2](#_Toc487196756)

[1. Einführung und Ziele / Fachliche Anforderungen 3](#_Toc487196757)

[1.1 Aufgabenstellung 3](#_Toc487196758)

[1.2 Qualitätsziele 3](#_Toc487196759)

[1.3 Stakeholder 3](#_Toc487196760)

[2. Randbedingungen 4](#_Toc487196761)

[2.1 Technische Randbedingungen 4](#_Toc487196762)

[3. Kontext 5](#_Toc487196763)

[3.1 Externe Schnittstellen 6](#_Toc487196764)

[3.1.1 Externe Schnittstelle 1 6](#_Toc487196765)

[3.1.2 Externe Schnittstelle 2 6](#_Toc487196766)

[3.1.4 Externe Schnittstelle 3 7](#_Toc487196767)

[4. Lösungsstrategie 8](#_Toc487196768)

[5. Bausteinsicht 9](#_Toc487196769)

[5.1 Ebene 1 9](#_Toc487196770)

[5.1.1 Bausteinname 1 (BlackBox-Beschreibung) 9](#_Toc487196771)

[5.2 Ebene 2 10](#_Toc487196772)

[5.2.1 Bausteinname 1 (Whitebox-Beschreibung) 10](#_Toc487196773)

[6. Laufzeitsicht 11](#_Toc487196774)

[7. Datensicht 12](#_Toc487196775)

[8. Verteilungssicht 13](#_Toc487196776)

[9. Konzepte 14](#_Toc487196777)

[9.1 Frameworks und Entwurfsmuster 14](#_Toc487196778)

[9.1.1 Frameworks 14](#_Toc487196779)

[9.1.2 Entwurfsmuster 14](#_Toc487196780)

[9.2 Persistenz 14](#_Toc487196781)

[9.3 Benutzungsoberfläche 14](#_Toc487196782)

[10. Qualitätsszenarien 15](#_Toc487196783)

[10.1 Bewertungsszenarien 15](#_Toc487196784)

[11. Glossar 16](#_Toc487196785)

# Einführung und Ziele / Fachliche Anforderungen

## Aufgabenstellung

Erstellung einer Wetterstation als prototypische Anwendungsentwicklung mit Java

Technologie.

## Qualitätsziele

* Entwicklung einer Wetterstation, die Wetterdaten wie Temperatur, Luftdruck, Helligkeit und Luftfeuchte misst und darstellt.
* Eine Verfügbarkeit von 24 Stunden am Tag, sieben Tage die Woche.
* Eine Schulungszeit von unter 10 Minuten die Benutzer benötigen, um eigenständig mit der Software arbeiten zu können
* Eine größtmögliche Testbarkeit ist zu gewährleisten um den Testaufwand so gering wie möglich zu halten.

## Stakeholder

|  |  |  |
| --- | --- | --- |
| **Name** | **Rolle** | **Definition** |
| Tobias Gilnhammer | Projektleiter | Koordination des Projektteammitglieder |
| Behnam Tajedini und Matthias Jeschke | Projektauftraggeber | Beauftragung und Unterstützung des Projektteams |
| Sinan Özturan | Designer | Designt die Masken am Desktop sowie am LCD-Bildschirm |
| Tarek Lutz | Entwickler | Entwickelt die Software für die Wetterstation |
| Tobias Gilnhammer, Tarek Lutz, Sinan Özturan und Sonja Wasenmiller | Tester | Die Tester testen die Software auf Fehler und geben Verbesserungsvorschläge ab. |
| Behnam Tajedini und Matthias Jeschke | Anwender | Nutzt die Anwendung |
| Tarek Lutz Tobias Gilnhammer Sinan Özturan Sonja Wasenmiller | Projektteammitglieder | Bearbeiten das Projekt, Erstellen die Anwendung |

# Randbedingungen

Im Wiki unter <https://github.com/matthjes/FISI-WS/wiki> sind alle Informationen eingetragen die das Projekt vereinfachen und an die sich grob gehalten werden soll. Eigeninitiativen der Entwickler zur Erfüllung der Anforderungen sind gerne gesehen.

## Technische Randbedingungen

|  |  |
| --- | --- |
| Hardware-Vorgaben | |
| Wetterstation von Tinkerforge | vollwertige Open Source Wetterstation. Sie misst Temperatur, Luftfeuchtigkeit, Luftdruck und Helligkeit mit hoher Präzision. |
| Raspberry Pi | Mini Computer auf dem ein Deamon zum Abgreifen der Wetterdaten von der Wetterstation läuft. |
| Windows/ OSX Rechner | Betrieb der Lösung auf einem Marktüblichen Standardrechner mit Microsoft Windows oder Apple OSX als Betriebssystem. |
| Software-Vorgaben | |
| Fremdsoftware frei verfügbar | Falls zur Lösung Fremdsoftware hinzugezogen wird (z.B. grafisches Frontend), sollte diese idealerweise frei verfügbar und kostenlos sein. |
| Konfigurations- und Versionsverwaltung | Konfigurations- und Versionsverwaltung über Git und Github. |
| Programmiervorgaben | |
| Backend | Erstellung von Java-Quelltexten in der IDE IntelliJ. |
| Frontend | Erstellung von Masken mittels JavaFX und SceneBuilder |

1. Absprache mit Entwickler

|  |  |
| --- | --- |
| Analyse- und Entwurfsmethoden | Objektorientierte oder strukturierte Methoden? |
| Datenstrukturen | Vorgaben für bestimmte Datenstrukturen, Schnittstellen zu bestehenden Datenbanken oder Dateien |
| Programmierschnittstellen | Schnittstellen zu bestehenden Programmen |
| Programmiervorgaben | Programmierkonventionen, fester Programmaufbau |
| Technische Kommunikation | Synchron oder asynchron, Protokolle |

# Kontext

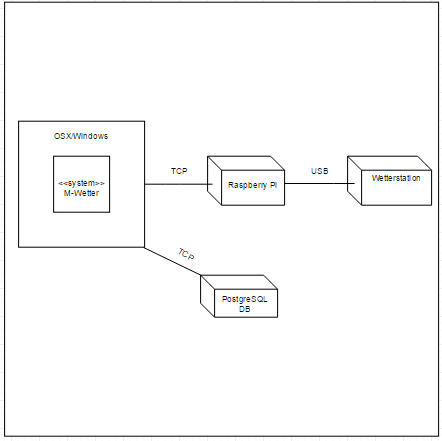


Bild: Technische Interaktion des Systems mit den anderen Komponenten.

Die Wetterstation ist mittels USB mit einem Raspberry Pi verbunden. Auf diesem läuft ein   
Deamon mit dem die Wetterdaten von der Wetterstation abgegriffen werden. Diese Wetterdaten werden lokal auf einer PostgreSQL Datenbank gespeichert und werden von der Anwendung aus der Datenbank ausgelesen. Anschließend werden die Wetterdaten auf dem Desktop und auf dem LCD Bildschirm der Wetterstation angezeigt.

## Externe Schnittstellen

### Externe Schnittstelle 1

#### Identifikation der Schnittstelle

|  |  |
| --- | --- |
| Name / Bezeichnung der Schnittstelle | RaspWetter |
| Version | 0.1 |
| Typ der Schnittstelle | Protokollorientiert |
| Anforderungen an die Schnittstelle | Zuverlässigkeit, Korrektheit der Datenübertragung |
| Ablauf der Schnittstelle | Wetterdaten werden von der Wetterstation auf den Raspberry übertragen |

### Externe Schnittstelle 2

#### Identifikation der Schnittstelle

|  |  |
| --- | --- |
| Name / Bezeichnung der Schnittstelle | <Name der Schnittstelle> |
| Version | 0.1 |
| Typ der Schnittstelle | Protokollorientiert |
| Anforderungen an die Schnittstelle | Zuverlässigkeit, Korrektheit der Datenübertragung |
| Ablauf der Schnittstelle | Wetterdaten werden von einem Brick-Deamon auf dem Raspberry zur Verfügung gestellt und von einem Rechner abgerufen. |

### Externe Schnittstelle 3

#### Identifikation der Schnittstelle

|  |  |
| --- | --- |
| Name / Bezeichnung der Schnittstelle | DB |
| Version | 0.1 |
| Typ der Schnittstelle | Protokollorientiert |
| Anforderungen an die Schnittstelle | Zuverlässigkeit, Korrektheit der Datenübertragung |
| Ablauf der Schnittstelle | Datenaustausch zwischen Datenbank und Client |

# Lösungsstrategie

Es gibt 2 Anwendungen. Die erste Anwendung dient dazu, die Wetterdaten auszulesen und in eine Datenbank zu speichern. Die Anwendung wird gestartet und verbindet sich mit dem Deamon der auf dem Raspberry läuft und ruft die Daten der Wetterstation ab. Der Deamon speichert diese Wetterdaten zwischen, verbindet sich mit der PostgreSQl Datenbank und schreibt die Daten in die Datenbank.

Die zweite Anwendung ist für den Benutzer gedacht. Der Anwender startet die Anwendung, welche sich dann mit der Datenbank verbindet und die erforderlichen Daten abruft. Anschließend werden die Daten in einer GUI ausgegeben.

# Bausteinsicht

Inhalt

Statische Zerlegung des Systems in Bausteine (Module, Komponenten, Subsysteme, Teilsysteme, Klassen, Interfaces, Pakete, Bibliotheken, Frameworks, Schichten, Partitionen, Tiers, Funktionen, Makros, Operationen, Datenstrukturen...) sowie deren Beziehungen.

Form

Die Bausteinsicht ist eine hierarchische Sammlung von BlackBox- und White-Box-Beschreibungen

## Ebene 1

Das Überblicksbild zeigt das Innenleben des Gesamtsystems in Form der Bausteine 1 - n, sowie deren Zusammenhänge und Abhängigkeiten.

Sinnvoll sind hier auch Beschreibungen der wichtige Begründungen, die zu dieser Struktur führen, insbesondere die Beschreibung der Abhängigkeiten (Beziehungen) zwischen den Bausteinen dieser Ebene.

### Bausteinname 1 (BlackBox-Beschreibung)

* **Zweck / Verantwortlichkeit:**
* **Schnittstelle(n):**
* **Erfüllte Anforderungen**

## Ebene 2

### Bausteinname 1 (Whitebox-Beschreibung)

#### Bausteinname 1.1 (BlackBox Beschreibung)

1. Zweck / Verantwortlichkeit:
2. Schnittstelle(n):
3. Erfüllte Anforderungen:

# 

# Laufzeitsicht

Inhalt

Diese Sicht beschreibt, wie sich die Bausteine des Systems als Laufzeitelemente (Prozesse, Tasks, Activities, Threads, ...) verhalten und wie sie zusammenarbeiten.

Anmerkung: Kriterium für die Auswahl der möglichen Szenarien (d.h. Abläufe) des Systems ist deren *Architekturrelevanz.* Es geht nicht darum, möglichst viele Abläufe darzustellen, sondern eine angemessene Auswahl zu dokumentieren.

Kandidaten sind:

1. Die wichtigsten 3-5 Anwendungsfälle
2. Systemstart
3. Das Verhalten an den wichtigsten externen Schnittstellen
4. Das Verhalten in den wichtigsten Fehlerfällen

Form

Dokumentieren Sie die ausgesuchten Laufzeitszenarien mit UML-Sequenz-, Aktivitäts-, oder Kommunikationsdiagrammen.

# Datensicht

Welche Kategorien von Daten müssen gespeichert werden?

Wo werden die einzelnen Daten gespeichert?

# 

# Verteilungssicht

Inhalt

Diese Sicht beschreibt, in welcher Umgebung das System abläuft. Sie beschreiben die geographische Verteilung Ihres Systems oder die Struktur der Hardwarekomponenten, auf denen die Software abläuft. wSie dokumentiert Rechner, Prozessoren, Netztopologien und Kanäle, sowie sonstige Bestandteile der physischen Systemumgebung. Die Verteilungssicht zeigt das System aus Betreibersicht.

Zeigen Sie in dieser Sicht auch, wie die Bausteine des Systems zu *Verteilungsartefakten* zusammengefasst oder –gebaut werden (engl. deployment artifacts oder deployment units).

Form

Die UML stellt mit Verteilungsdiagrammen (Deployment diagrams) eine Diagrammart zur Verfügung, um diese Sicht auszudrücken. Nutzen Sie diese, evtl. auch geschachtelt, wenn Ihre Verteilungsstruktur es verlangt. (Das oberste Deployment- Diagramm sollte bereits in Ihrer Kontextsicht enthalten sein mit Ihrer Infrastruktur als EINE Black-Box. Jetzt zoomen Sie in diese Infrastruktur mit weiteren Deployment- Diagrammen hinein.

Andere Diagramme Ihrer Hardware-Kollegen, die Knoten und Kanäle darstellen sind hier ebenfalls einsetzbar. Abstrahieren Sie aber auf die Aspekte, die für die Software-Verteilung relevant sind.

# Konzepte

## Frameworks und Entwurfsmuster

### Frameworks

Welche Frameworks werden im Anwendungssystem verwendet?

Welche Funktionen decken sie jeweils ab?

## Persistenz

Persistenz (Dauerhaftigkeit, Beständigkeit) bedeutet, Daten aus dem (flüchtigen) Hauptspeicher auf ein beständiges Medium (und wieder zurück) zu bringen.

## Benutzungsoberfläche

Es ist eine durch JAVAFX realisierte Benutzeroberfläche die durch wechseln der verschiedenen Tabs (Textanzeige, Graph) verändert werden kann.

# Qualitätsszenarien

Dieses Kapitel fasst alles zusammen, was Sie zur systematischen Bewertung Ihrer Architektur gegen vorgegebene Qualitätsziele benötigen.

## Bewertungsszenarien

Inhalt

Szenarien beschreiben, was beim Eintreffen eines Stimulus auf ein System in bestimmten Situationen geschieht. Sie charakterisieren damit das Zusammenspiel von Stakeholdern mit dem System. Szenarien operationalisieren Qualitätsmerkmale und machen sie messbar.

Wesentlich für die meisten Software-Architekten sind zwei Arten von Szenarien:

1. Nutzungsszenarien (auch genannt *Anwendungs- oder Anwendungsfallszenarien*) beschreiben, wie das System zur Laufzeit auf einen bestimmten Auslöser reagieren soll. Hierunter fallen auch Szenarien zur Beschreibung von Effizienz oder Performance. Beispiel: Das System beantwortet eine Benutzeranfrage innerhalb einer Sekunde.
2. Änderungsszenarien beschreiben eine Modifikation des Systems oder seiner unmittelbarer Umgebung. Beispiel: Eine zusätzliche Funktionalität wird implementiert oder die Anforderung an ein Qualitätsmerkmal ändert sich.



**Abbildung: Schematische Darstellung von Szenarien (nach [Bass+03])**

Szenarien bestehen aus folgenden wesentlichen Teilen (hier zitiert aus [Starke05], die ursprüngliche Gliederung stammt aus [Bass+03]):

1. Auslöser (*stimulus*): beschreibt eine spezifische Zusammenarbeit des (auslösenden) Stakeholders mit dem System. Beispiele: Ein Benutzer ruft eine Funktion auf, ein Entwickler programmiert eine Erweiterung, ein Administrator installiert oder konfiguriert das System.
2. Quelle des Auslösers (*source*): beschreibt, woher der Auslöser kommt. Beispiele: intern oder extern, Benutzer, Betreiber, Angreifer, Manager.
3. Umgebung (*environment*): beschreibt den Zustand des Systems zum Zeitpunkt des Auslösers. Befindet sich das System unter Normal- oder Höchstlast? Ist die Datenbank verfügbar oder nicht? Sind Benutzer online oder nicht? Hier sollten Sie alle Bedingungen beschreiben, die für das Verständnis des Szenarios wichtig sind.
4. Systembestandteil (*artifact*): beschreibt, welcher Bestandteil des Systems vom Auslöser betroffen ist. Beispiele: Gesamtsystem, Datenbank, Webserver.
5. Antwort (*response*): beschreibt wie das System durch seine Architektur auf den Auslöser reagiert. Wird die vom Benutzer aufgerufene Funktion ausgeführt? Wie lange benötigt der Entwickler zur Programmierung? Welche Systemteile sind von Installation/Konfiguration betroffen?
6. Antwortmetrik (*response measure*): beschreibt, wie die Antwort gemessen oder bewertet werden kann. Beispiele: Ausfallzeit in Stunden, Korrektheit Ja/Nein, Änderungszeit in Personentagen, Reaktionszeit in Sekunden.

Form

Entweder tabellarisch oder als Freitext. Sie sollten die Bestandteile (Quelle, Umgebung, Systembestandteil, Antwort, Antwortmetrik) explizt kenntlich machen.

# 

# Glossar

Inhalt

Die wichtigsten Begriffe der Software-Architektur in alphabetischer Reihenfolge

|  |  |
| --- | --- |
| Begriff | Definition |
| IDE | Integrated Development Envirement (Integrierte Entwicklungsumgebung) |
|  |  |