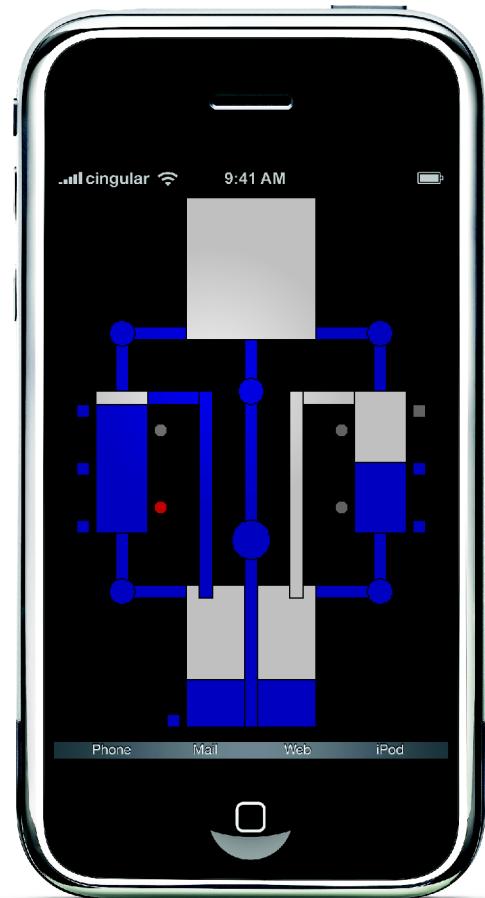


# iPump



iPump your Projektpraktikum

by

David Brühlmeier

Alex Müri

Michael Bhend

Marcel Suter

# Inhaltsverzeichnis

1.Einführung.....	4
2.Produkte-Karton.....	5
3.Zeitplan.....	6
4.Fragen und Ergänzungen zum Pflichtenheft.....	7
5.MindMap.....	8
6.Anwendungsfallmodell.....	9
7.Anwendungsfallbeschreibungen.....	10
7.1.Parameter eingeben / Programm beenden.....	10
7.2.Resetknopf drücken.....	12
7.3.Logging auslesen / Datei auslesen.....	14
7.4.Logging auslesen / Datenbank auslesen.....	14
7.5.Logging löschen / Datei löschen.....	14
7.7.Logging auslesen / Datenbank löschen.....	15
8.Szenarien.....	16
9.OO-Synthese.....	19
9.1.Vorgehen nach Godet.....	19
9.2.Einfärben der Begriffe.....	19
9.3.Erstellen einer Tabelle der Designelemente.....	21
10.Prototypen.....	23
10.1.Prototyp für Initialisierung.....	23
10.2.UI-Prototyp.....	24
11.Fachklassen-Diagramm und Beschreibung.....	25
11.1.Fachklassendiagramm.....	25
11.3.Beschreibung.....	26
12.Detaildesign der Sensoren mit Fehlerkorrektur.....	27
12.1.Logik.....	27
12.2.Whiteboxtest (Codebeispiel).....	28
12.3.Whiteboxtest (Consolen Printout).....	29
13.Software-Design.....	30
13.1.Systemklassendiagramm.....	30
13.2.Zustandsdiagramme.....	31
13.2.1Klasse BottomTank.....	31
13.2.2Klasse Lamp.....	32
13.2.3Klasse SideTank.....	33
13.2.4Klasse TankControl.....	34
13.3.Sequenzdiagramm.....	35
13.4.Attribut- und Operationenbeschreibung.....	36
14.Testplan.....	53
14.1.Heading2.....	53
14.2.Heading2.....	53
14.2.1Heading3.....	53
15.Installations- und Betriebsanleitung.....	54
16.Timingdiagramm.....	55
17.Gemachte Erfahrungen.....	56
17.1.Dokumentation mit OpenOffice.....	56
17.2.Codeverwaltung mit TortoiseSVN und GoogleCode.....	56
17.3.Designtool Enterprise Architect.....	56
17.4.Design-Entscheide festhalten.....	56
17.5.Codegenerierung mit EAP.....	56
17.6.Codierung.....	56
18.Glossar.....	58

## **Illustrationsverzeichnis**

Illustration 1: Projektteam (v.l.n.r: Marcel Suter, Alex Müri, David Brühlmeier, Michael Bhend.....	5
Illustration 2: Produktekartion.....	6
Illustration 3: Zeitplan.....	8
Illustration 4: Mindmap aus dem Pflichtenheft.....	10
Illustration 5: Anwendungsfallmodell.....	11
Illustration 6: Szenario 1.....	18
Illustration 7: Szenario 2.....	19
Illustration 8: Szenario 3.....	20
Illustration 9: OO-Synthese mittels Sammeln von Begriffen.....	22
Illustration 10: UI-Prototyp mit neuem Abbruchkriterium.....	26
Illustration 11: Fachklassendiagramm.....	27
Illustration 12: Systemklassendiagramm.....	32
Illustration 13: Zustandsdiagramm für BottomTank.....	33
Illustration 14: Zustandsdiagramm für Lamp.....	34
Illustration 15: Zustandsdiagramm für SideTank.....	35
Illustration 16: Zustandsdiagramm für TankControl.....	36
Illustration 17: Sequenzdiagramm für Applikationsstart.....	37
Illustration 18: Timing-Diagramm.....	58

# 1. Einführung

iPump ist eine Pumpensteuerung, die im Rahmen eines Projektpraktikums an der Softwareschule Bern entwickelt wurde. Das Projekt basiert auf den von den Dozenten abgegebenen Unterlagen.

Die vorliegende Dokumentation beschreibt den Entwicklungsprozess für die Steuerungssoftware.



*Illustration 1: Projektteam (v.l.n.r: Marcel Suter, Alex Müri, David Brühlmeier, Michael Bhend*

## 2. Produkte-Karton

Unser Produktekarton ist ein DVD-Case. Unten finden Sie eine verkleinerte Version.

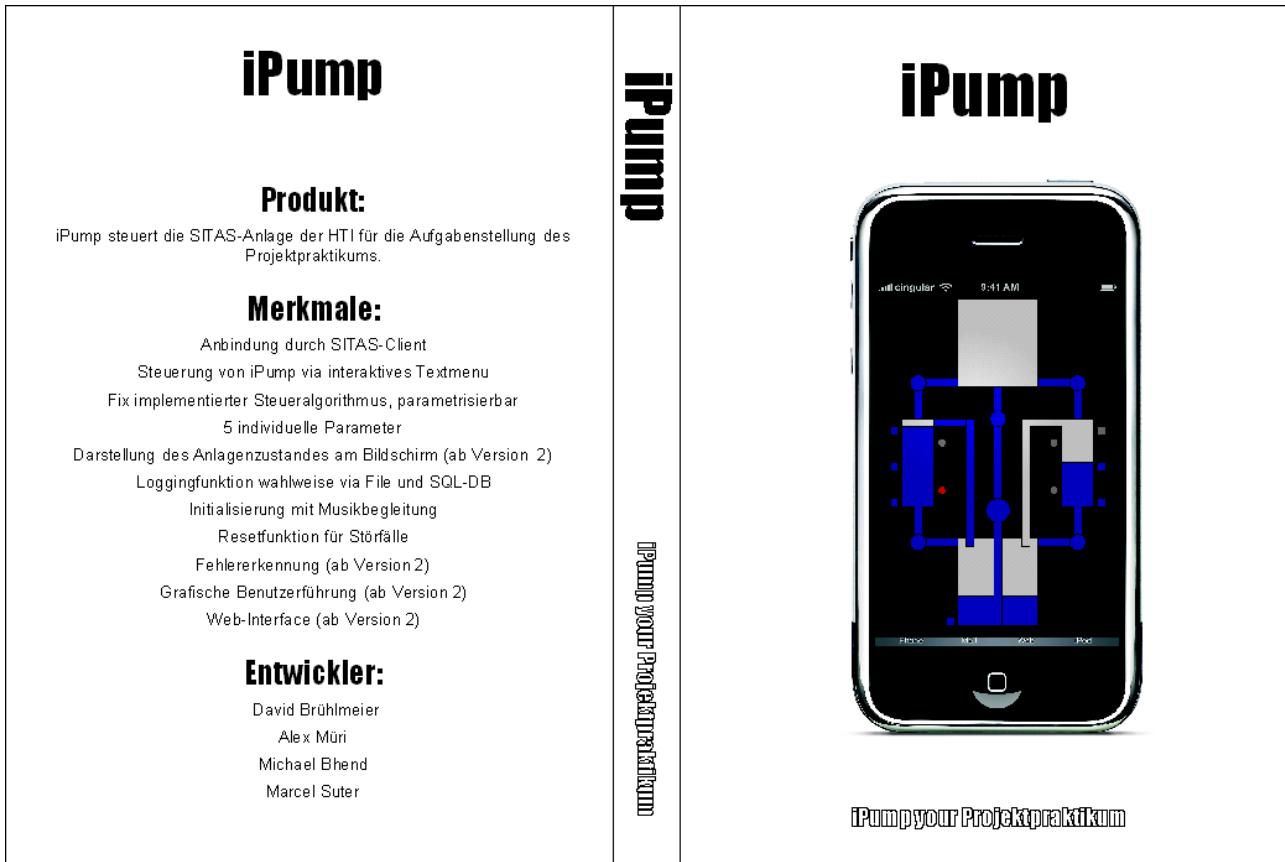


Illustration 2: Produktekartion

### 3. Zeitplan

Unser Zeitplan enthält eine Übersicht über die Zeiten, die wir in der Schule zur Verfügung haben, sowie die zu erledigenden Arbeiten und zu erstellenden Dokumente. Wir konnten uns erstaunlich gut an diesen Zeitplan halten.

Zeitplan Projekt SITAS Gruppe 1

	16.01.07	17.01.07	18.01.07	19.01.07	20.01.07	21.01.07	22.01.07	23.01.07	24.01.07	25.01.07	26.01.07	27.01.07	28.01.07	29.01.07	30.01.07	31.01.07	01.02.07	02.02.07	03.02.07	04.02.07	05.02.07	06.02.07	07.02.07	08.02.07	
	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	
Schule																									
Grob Analyse																									
OOA																									
OOS																									
OOD																									
Programmieren																									
Testen																									
Präsentation																									
Arbeiten																									
Grob-Analyse																									
OOA																									
OOS																									
OOD																									
Programmieren																									
Testen																									
Doku zusammenstellen																									
Präsentation vorbereiten																									
Prototyp Statemachine																									
Prototyp JDBC																									
Prototyp GUI																									
Dokumente																									
OOA	Mind-Map																								
	Fachliches Glossar																								
	Produktkarton																								
	Anwendungsfall-Modell																								
	Anwendungsfall-Beschreibungen																								
	Szenarien																								
	Timing-Diagramm																								
OOS	OO-Synthese																								
	Fachklassen-Diagramm																								
	Beschreibung der Fachklassen																								
OOD	Systemklassen-Diagramm																								
	Zustands-Diagramme																								
	Aktivitäts-Diagramme																								
	ev. Sequenz-Diagramme																								
	Attributs- und Operations-Beschr.																								
	Betriebs- und Installationsanweisung																								
	Java Source-Code																								
	Java-Doc (CD)																								
	Test-Plan (Vorgaben und erw. Resu.)																								
	Zeitplan																								
	Erfahrungen und Lehren																								

*Illustration 3: Zeitplan*

## 4. Fragen und Ergänzungen zum Pflichtenheft

Bei der Arbeit mit iPump haben sich die folgenden Fragen gestellt, die wir zusätzlich spezifiziert haben, mit Rücksprache mit dem Auftraggeber.

### Füllung des linken Tanks beim letzten Zyklus

Falls der linke Tank bei der letzten Füllung des rechten Tanks noch nicht voll ist (z.B. Ventil fast zu), wird dies abgewartet. Erst wenn beide Tanks voll sind, gilt der Zyklus als abgeschlossen.

### UI

Wir haben das UI als textbasiertes Menu implementiert. Zusätzlich wurde die Abbruchbedingung als Ja/Nein-Frage implementiert. Es wird eine Konformitätsprüfung der Benutzereingaben vorgenommen; wenn die Eingabe unsinnig ist, wird sie zurückgewiesen.

### Datenbank

Wir verwenden die Standard-ASA Datenbank aus dem SQL-Unterricht.

### Logging

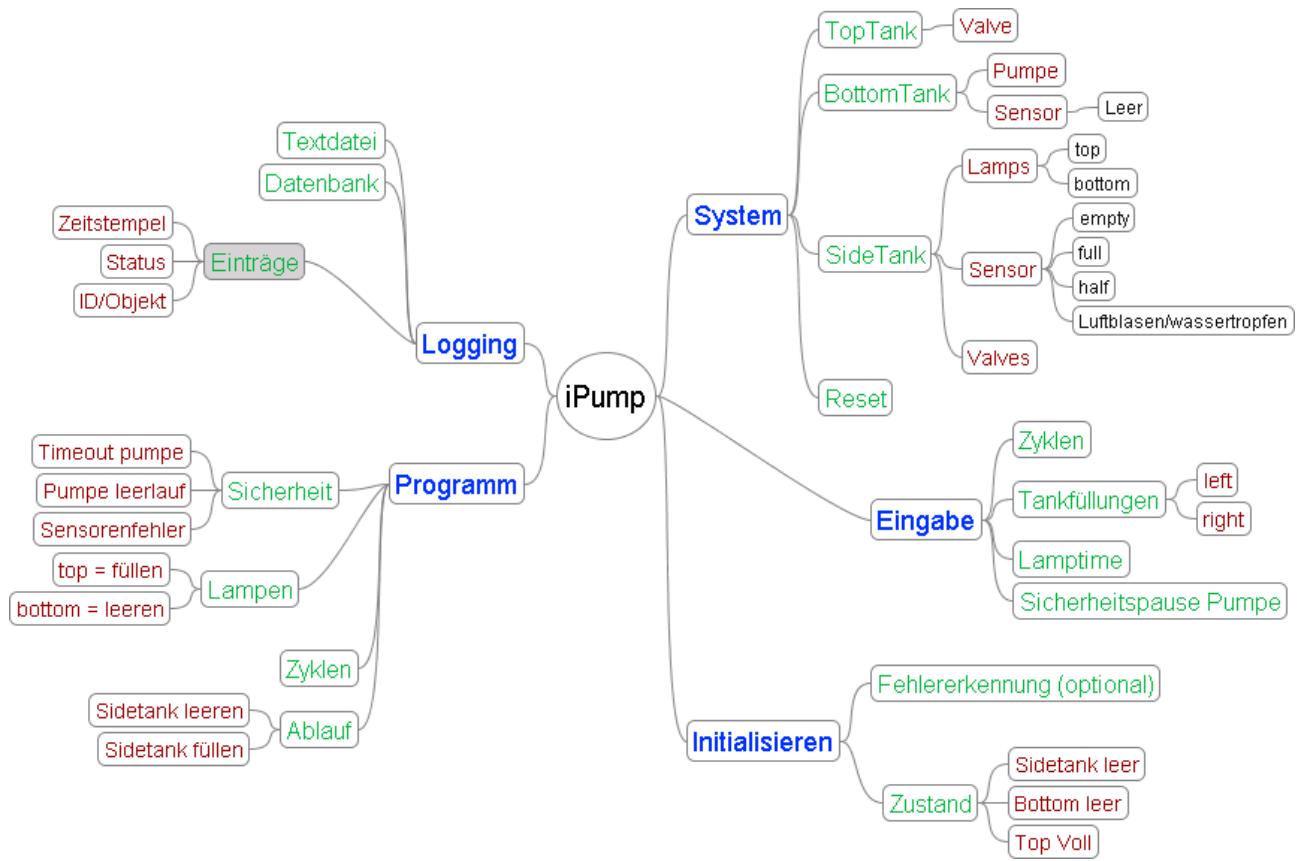
Es werden der Zeitstempel, ein Nachrichtentyp, eine Herkunft und ein String abgelegt.

### Ini-File

Um den Debuggingvorgang zu erleichtern, verwenden wir ein INI-File, das die Parameter und verschiedene Settings beinhaltet. Unter anderem kann auch das GUI umgangen werden.

## 5. MindMap

Das MindMap unseres Projektes beinhaltet die wesentlichen Aspekte des Pflichtenheftes.



*Illustration 4: Mindmap aus dem Pflichtenheft*

## 6. Anwendungsfallmodell

Unser Anwendungsfallmodell enthält die vom Pflichtenheft geforderten Anwendungsfälle. Die zugehörigen Beschreibungen befinden sich im nächsten Kapitel.

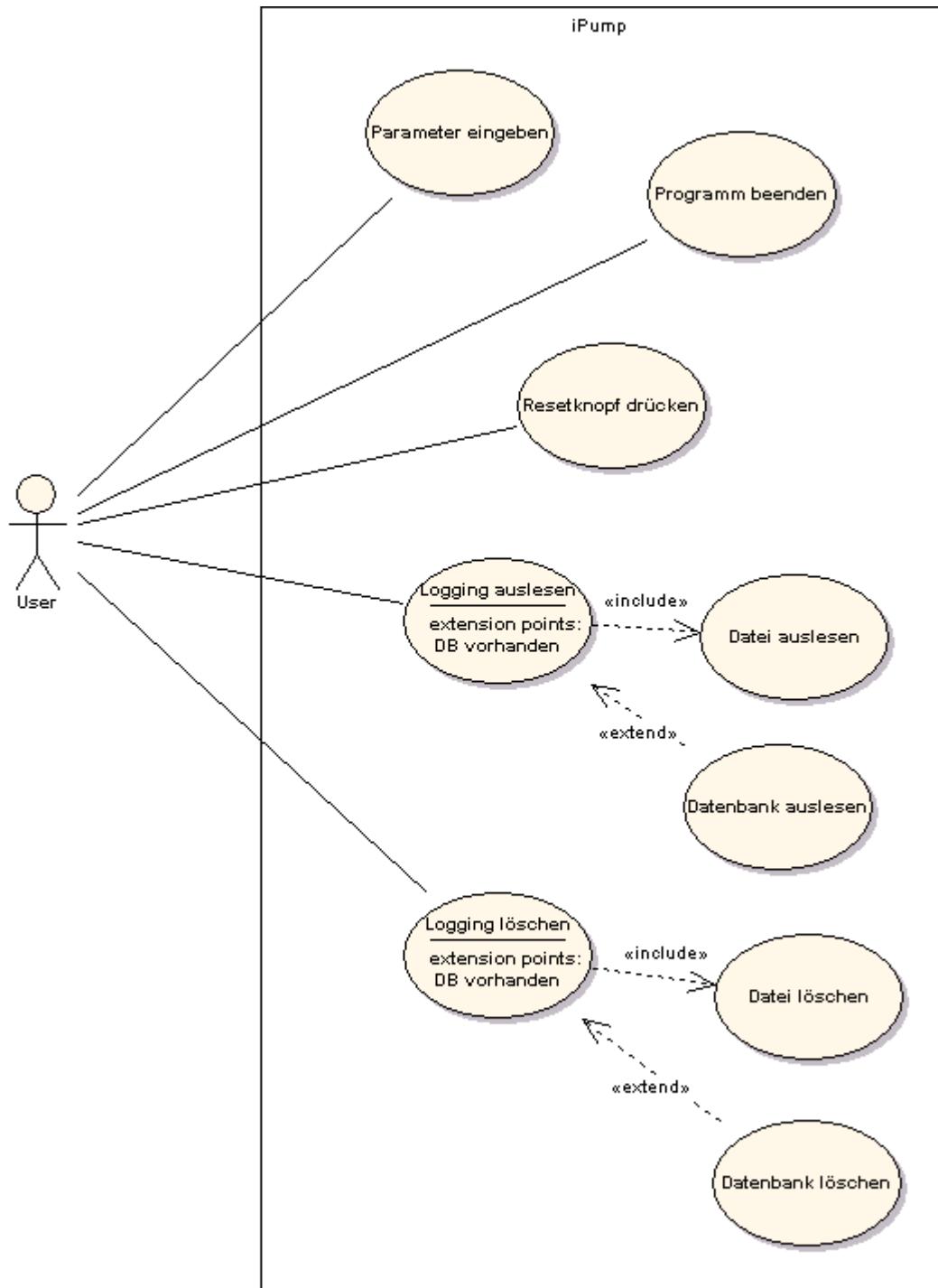


Illustration 5: Anwendungsfallmodell

## 7. Anwendungsfallbeschreibungen

Hier finden Sie die Beschreibungen der Anwendungsfälle. Wir haben nur die für uns wichtigen hier genauer beschrieben und je nach Wichtigkeit sind mehrere Detailstufen vorhanden.

### 7.1. Parameter eingeben / Programm beenden

Geschäftsanwendungsfall	
Name des AWF	<b>Parameter eingeben / Programm beenden</b>
Kurzbeschreibung	Parameter werden in textbasiertem User Interface (Konsole) eingegeben.
Auslöser/Motivation	Anwender will iPump starten
Ergebnis	Parameter eingelesen. Pumpe wird gestartet oder Programm wird beendet.
Akteure	Anwender

Essenzieller Geschäftsanwendungsfall	
Name des AWF	<b>Parameter eingeben / Programm beenden</b>
Kurzbeschreibung	Parameter werden in Textbasierten User Interface (Console) eingegeben.
Auslöser/Motivation	Anwender will iPump starten
Ergebnis	Parameter eingelesen. Pumpe wird gestartet oder Programm wird beendet.
Akteure	Anwender
Eingehende Information	<ul style="list-style-type: none"><li>• Anzahl Füllungs-/Leerungs-Zyklen für den linken Tank [1..n]</li><li>• Anzahl Füllungs-/Leerungs-Zyklen für den rechten Tank [1..n]</li><li>• Anzahl Abläufe (Zyklen des linken und rechten Tanks)</li><li>• Leuchtdauer der Lampen in Sekunden [1..n]</li><li>• Ausschaltzeit der Pumpe, nachdem der untere Tank leer geworden ist, in Sekunden [1..n]</li></ul>
Vorbedingungen	Programm gestartet. Eingabeaufforderung erscheint auf der Konsole.
Nachbedingungen	Alle Parameter richtig eingegeben.
Essenzielle Schritte	Alle Parameter nacheinander eingegeben. Parameter werden auf Gültigkeit und Bereich überprüft.

Implementations-Anwendungsfall	
Name des AWF	<b>Parameter eingeben / Programm beenden</b>
Kurzbeschreibung	Parameter werden in Textbasierten User Interface (Console) eingegeben.
Auslöser/Motivation	Anwender will iPump starten oder beenden
Ergebnis	Parameter eingelesen. Pumpe wird gestartet oder Programm wird beendet.
Akteure	Anwender
Eingehende Information	<ul style="list-style-type: none"> <li>• Anzahl Füllungs-/Leerungs-Zyklen für den linken Tank [1..n]</li> <li>• Anzahl Füllungs-/Leerungs-Zyklen für den rechten Tank [1..n]</li> <li>• Anzahl Abläufe (Zyklen des linken und rechten Tanks) [1..n; resp. 0 = Programmende]</li> <li>• Leuchtdauer der Lampen in Sekunden [1..n]</li> <li>• Ausschaltzeit der Pumpe, nachdem der untere Tank leer geworden ist, in Sekunden [1..n]</li> </ul>
Vorbedingungen	Programm gestartet. Eingabeaufforderung erscheint auf der Console.
Nachbedingungen	Alle Parameter richtig eingegeben. Ablauf fertig durchgelaufen. Parameter geloggt.
Ergebnis (positiver Fall)	Ablauf wird gestartete oder Programm wird beendet
Ergebnis (negativer Fall)	Gibt es nicht. Bei einem falsch eingegeben Wert wird die Eingabe solange wiederholt bis die Eingabe richtig ist. Ist die iPump einmal gestartet kann sie mit dem Reset Knopf wieder abgebrochen.
Ablauf	Jeder der Parameter wird nacheinander abgefragt. Nach der Eingabe wird er auf Gültigkeit geprüft. Nach Eingabe des letzten Parameters wird die Ablauf gestartet.
Variationen	Wenn der Benutzer „Nein“ auf die Frage zur Programmbeendung eingibt, wird das Programm beendet.
Ausnahmen, Fehler	Eingabewerte müssen auf Richtigkeit geprüft werden.
Regeln	Keine
Invariaten	Keine
Allgemeine Systemdienste	PC mit DOS-Konsole. Angeschlossenes SITAS-Model oder Simulator
Fremdspezifikationen	Keine
Nicht funktionale Anforderungen	Keine
Verweis auf Diagramme	Keine
Dialogbeispiele (Prototyp)	Keine
Ansprechpartner	Keine
Mögliche Erweiterungen	Keine
Offene Fragen	Keine
Änderungs-History	Keine

## 7.2. Resetknopf drücken

Geschäftsanwendungsfall	
Name des AWF	<b>Resetknopf drücken</b>
Kurzbeschreibung	Auf dem SITAS-Modell oder Simulator wird der Resetknopf gedrückt.
Auslöser/Motivation	Der Ablauf möchte vom Anwender schnell beendet werden.
Ergebnis	Ablauf wird beendet. Programm gibt auf der Konsole eine Fehlermeldung aus.
Akteure	Anwender

Essenzieller Geschäftsanwendungsfall	
Name des AWF	<b>Resetknopf drücken</b>
Kurzbeschreibung	Auf dem SITAS-Modell oder Simulator wird der Resetknopf gedrückt.
Auslöser/Motivation	Der Ablauf möchte vom Anwender schnell beendet werden.
Ergebnis	Ablauf wird beendet. Programm gibt auf der Konsole eine Fehlermeldung aus.
Akteure	Anwender
Eingehende Information	Die Methode isReset() des liefert TRUE
Vorbedingungen	Programm ist am Laufen
Nachbedingungen	Alle Ventile geschlossen. Pumpe gestoppt. Fehlermeldung ausgegeben.
Essenzielle Schritte	<ol style="list-style-type: none"><li>1. Pumpe stoppen</li><li>2. Ventile schliessen</li><li>3. Fehlermeldung auf Konsole ausgeben.</li></ol>

Implementations-Anwendungsfall	
Name des AWF	<b>Resetknopf drücken</b>
Kurzbeschreibung	Auf dem SITAS-Modell oder Simulator wird der Resetknopf gedrückt.
Auslöser/Motivation	Der Ablauf möchte vom Anwender schnell beendet werden.
Ergebnis	Ablauf wird beendet. Programm gibt auf der Konsole eine Fehlermeldung aus.
Akteure	Anwender
Eingehende Information	Die Methode isReset() des liefert TRUE
Vorbedingungen	Programm ist am Laufen
Nachbedingungen	Alle Ventile geschlossen. Pumpe gestoppt.
Ergebnis (positiver Fall)	Pumpe wird gestartete oder Programm wird beendet
Ergebnis (negativer Fall)	Gibt es nicht. Ist eine Steuerung und keine Regelung.
Ablauf	<ol style="list-style-type: none"> <li>1. Pumpe stoppen</li> <li>2. Ventile schliessen</li> <li>3. Fehlermeldung auf Konsole ausgeben.</li> </ol>
Variationen	keine
Ausnahmen, Fehler	keine
Regeln	Keine
Invariaten	Keine
Allgemeine Systemdienste	Methode isReset() der Klasse TankSystem. Steuermethoden für Ventile und Pumpe.
Fremdspezifikationen	Keine
Nicht funktionale Anforderungen	Keine
Verweis auf Diagramme	Keine
Dialogbeispiele (Prototyp)	Keine
Ansprechpartner	Keine
Mögliche Erweiterungen	Keine
Offene Fragen	Keine
Änderungs-History	Keine

### **7.3. Logging auslesen / Datei auslesen**

Geschäftsanwendungsfall	
Name des AWF	<b>Logging auslesen / Datei auslesen</b>
Kurzbeschreibung	Nach beenden des Ablaufs kann das Logging in der Datei ausgelesen werden. Das Auslesen geschieht ausserhalb des iPump Programms. Die Logdatei wird mit Windows aufgemacht.  Wird nicht implementiert.  Optional kann, wenn vor Abgabe des Projektes noch genügend Zeit besteht, ein Auslesen via Konsolenbedienung implementiert werden.
Auslöser/Motivation	Verschiedene Situationen für auf dem Modell
Ergebnis	Logging wird auf dem Bildschirm dargestellt.
Akteure	Anwender

### **7.4. Logging auslesen / Datenbank auslesen**

Geschäftsanwendungsfall	
Name des AWF	<b>Logging auslesen / Datenbank auslesen</b>
Kurzbeschreibung	Nach dem Beenden des Ablaufs kann das Logging aus der Datenbank ausgelesen werden.  Wird nicht implementiert. Das Auslesen aus der Datenbank geschieht direkt mit einem SQL-Utility Programm. Damit können auch detaillierte und selektive Abfragen gemacht werden.  Optional kann, wenn vor Abgabe des Projektes noch genügend Zeit besteht, ein Auslesen via Konsolenbedienung implementiert werden.
Auslöser/Motivation	Genauer Ablauf des Programms kontrollieren und nachvollziehen.
Ergebnis	Logging wird in Tabellenform dargestellt.
Akteure	Anwender

### **7.5. Logging löschen / Datei löschen**

Geschäftsanwendungsfall	
Name des AWF	<b>Logging löschen / Datei löschen</b>
Kurzbeschreibung	Nach dem Beenden des Ablaufs kann das Logging aus der Datei gelöscht werden.  Das Löschen der Datei geschieht ausserhalb des iPump Programms. Wird nicht implementiert.
Auslöser/Motivation	Logdatei möchte gelöscht werden.
Ergebnis	Logdatei ist im „Papierkorb“
Akteure	Anwender

## 7.6. Logging auslesen / Datenbank löschen

Geschäftsanwendungsfall	
Name des AWF	<b>Logging auslesen / Datenbank löschen</b>
Kurzbeschreibung	Nach dem Beenden des Ablaufs kann das Logging in der Datenbank gelöscht werden.  Wird nicht implementiert. Das Löschen der Daten in der Datenbank geschieht direkt mit einem SQL-Utility Programm. Optional kann, wenn vor Abgabe des Projektes noch genügend Zeit besteht, ein Löschen via Konsolenbedienung implementiert werden.
Auslöser/Motivation	Daten möchten gelöscht werden.
Ergebnis	Daten sind gelöscht.
Akteure	Anwender

## 8. Szenarien

Wir haben verschiedene Szenarien modelliert, indem wir den erwarteten Ablauf der Messages im Logfile modelliert haben. Dazu haben wir für die Parameter verschiedenen Werte angenommen und die Zeiten der jeweils erwarteten Meldungen berechnet.

Simulierte Log-File als Resultat des Szenarios			Beschreibung des Szenarios		
Zeit	Objekt	Nachricht	Regulärer Ablauf mit normalen Parametern.		
0.000	Controller	Start Init			
40.000	Controller	End Init			
40.001	Controller	Start Run 1			
40.002	System	Start Pumping			
56.669	BottomTankSensor	Off			
56.670	System	Stop Pumping			
56.671	Controller	Start Left Fill Cycle 1			
56.672	LeftTopValve	Open			
56.673	LeftTopLamp	On			
56.674	LeftBottomSensor	On			
61.672	LeftMiddleSensor	On			
64.673	LeftTopLamp	Off			
66.672	LeftTopSensor	On			
66.673	LeftTopValve	Close			
66.674	Controller	End Left Fill Cycle 1			
66.675	Controller	Start Left Empty Cycle 1			
66.676	LeftBottomValve	Open			
66.677	LeftBottomLamp	On			
67.076	LeftTopSensor	Off			
72.076	LeftMiddleSensor	Off			
74.077	LeftBottomLamp	Off			
77.076	LeftBottomSensor	Off			
77.077	LeftBottomValve	Close			
77.078	Controller	End Left Empty Cycle 1			
77.079	Controller	Start Left Fill Cycle 2			
77.080	LeftTopValve	Open			
77.081	LeftTopLamp	On			
77.280	LeftBottomSensor	On			
82.280	LeftMiddleSensor	On			
85.081	LeftTopLamp	Off			
87.280	LeftTopSensor	On			
87.281	LeftTopValve	Close			
87.282	Controller	End Left Fill Cycle 2			
87.283	Controller	Start Left Empty Cycle 2			
87.284	LeftBottomValve	Open			
87.285	LeftBottomLamp	On			
87.484	LeftTopSensor	Off			
92.484	LeftMiddleSensor	Off			
95.285	LeftBottomLamp	Off			
97.484	LeftBottomSensor	Off			
97.485	LeftBottomValve	Close			
97.486	Controller	End Left Empty Cycle 2			
97.487	Controller	Start Left Fill Cycle 3 (Last)			
97.488	LeftTopValve	Open			
97.489	LeftTopLamp	On			
97.688	LeftBottomSensor	On			
102.688	LeftMiddleSensor	On			
102.689	Controller	Start Right Fill Cycle 1			
102.690	RightTopValve	Open			
102.691	RightTopLamp	On			
102.890	RightBottomSensor	On			
105.489	LeftTopLamp	Off			
107.688	LeftTopSensor	On			
107.689	LeftTopValve	Close			
107.690	Controller	End Left Fill Cycle 3 (Last)			
107.890	RightMiddleSensor	On			
110.691	RightTopLamp	Off			
112.890	RightTopSensor	On			
112.891	RightTopValve	Close			
112.892	Controller	End Right Fill Cycle 1			
112.893	Controller	Start Right Empty Cycle 1			
112.894	RightBottomValve	Open			
112.895	RightBottomLamp	On			
113.094	RightTopSensor	Off			
118.094	RightMiddleSensor	Off			
120.895	RightBottomLamp	Off			
123.094	RightBottomSensor	Off			
123.095	RightBottomValve	Close			
123.096	Controller	End Right Empty Cycle 1			
123.097	Controller	Start Right Fill Cycle 2 (Last)			
123.098	RightTopValve	Open			
123.099	RightTopLamp	On			
123.298	RightBottomSensor	On			
128.298	RightMiddleSensor	On			
131.099	RightTopLamp	Off			
133.298	RightTopSensor	On			
133.299	Controller	Start Left Empty Cycle 3 (Last)			
133.299	RightTopValve	Close			
133.300	LeftBottomValve	Open			
133.300	Controller	End Right Fill Cycle 2 (Last)			
133.301	LeftBottomLamp	On			
133.301	Controller	Start Right Empty Cycle 2 (Last)			
133.302	RightBottomValve	Open			
133.303	RightBottomLamp	On			
133.500	LeftTopSensor	Off			
133.502	RightTopSensor	Off			
138.500	LeftMiddleSensor	Off			
138.502	RightMiddleSensor	Off			
141.301	LeftBottomLamp	Off			
141.303	RightBottomLamp	Off			
143.500	LeftBottomSensor	Off			
143.501	LeftBottomValve	Close			
143.502	Controller	End Left Empty Cycle 3 (Last)			
143.502	RightBottomSensor	Off			
143.503	RightBottomValve	Close			
143.504	Controller	End Right Empty Cycle 2 (Last)			

Illustration 6: Szenario 1

Simulierte Log-File als Resultat des Szenarios			Beschreibung des Szenarios		
Zeit	Objekt	Nachricht			
0.000	Controller	Start Init			
40.000	Controller	End Init			
40.001	Controller	Start Run 1			
40.002	System	Start Pumping			
56.669	BottomTankSensor	Off			
56.670	System	Stop Pumping			
56.671	Controller	Start Left Fill Cycle 1			
56.672	LeftTopValve	Open			
56.673	LeftTopLamp	On			
56.672	LeftBottomSensor	On			
64.673	LeftTopLamp	Off			
81.872	LeftMiddleSensor	On			
106.872	LeftTopSensor	On			
106.873	LeftTopValve	Close			
106.874	Controller	End Left Fill Cycle 1			
106.875	Controller	Start Left Empty Cycle 1			
106.876	LeftBottomValve	Open			
106.877	LeftBottomLamp	On			
107.076	LeftTopSensor	Off			
114.877	LeftBottomLamp	Off			
132.076	LeftMiddleSensor	Off			
157.076	LeftBottomSensor	Off			
157.077	LeftBottomValve	Close			
157.078	Controller	End Left Empty Cycle 1			
157.079	Controller	Start Left Fill Cycle 2			
157.080	LeftTopValve	Open			
157.081	LeftTopLamp	On			
157.280	LeftBottomSensor	On			
165.081	LeftTopLamp	Off			
182.280	LeftMiddleSensor	On			
207.280	LeftTopSensor	On			
207.281	LeftTopValve	Close			
207.282	Controller	End Left Fill Cycle 2			
207.283	Controller	Start Left Empty Cycle 2			
207.284	LeftBottomValve	Open			
207.285	LeftBottomLamp	On			
207.484	LeftTopSensor	Off			
215.283	LeftBottomLamp	Off			
232.484	LeftMiddleSensor	Off			
257.484	LeftBottomSensor	Off			
257.485	LeftBottomValve	Close			
257.486	Controller	End Left Empty Cycle 2			
257.487	Controller	Start Left Fill Cycle 3 (Last)			
257.488	LeftTopValve	Open			
257.489	LeftTopLamp	On			
257.688	LeftBottomSensor	On			
265.489	LeftTopLamp	Off			
282.688	LeftMiddleSensor	On			
282.689	Controller	Start Right Fill Cycle 1			
282.690	RightTopValve	Open			
282.691	RightTopLamp	On			
282.890	RightBottomSensor	On			
287.890	RightMiddleSensor	On			
290.691	RightTopLamp	Off			
292.890	RightTopSensor	On			
292.891	RightTopValve	Close			
292.892	Controller	End Right Fill Cycle 1			
292.893	Controller	Start Right Empty Cycle 1			
292.894	RightBottomValve	Open			
292.895	RightBottomLamp	On			
293.094	RightTopSensor	Off			
298.094	RightMiddleSensor	Off			
300.895	RightBottomLamp	Off			
303.094	RightBottomSensor	Off			
303.095	RightBottomValve	Close			
303.096	Controller	End Right Empty Cycle 1			
303.097	Controller	Start Right Fill Cycle 2 (Last)			
303.098	RightTopValve	Open			
303.099	RightTopLamp	On			
303.298	RightBottomSensor	On			
307.688	LeftTopSensor	On			
307.689	LeftTopValve	Close			
307.690	Controller	End Left Fill Cycle 3 (Last)			
308.298	RightMiddleSensor	On			
311.093	RightTopLamp	Off			
313.298	RightTopSensor	On			
313.299	Controller	Start Left Empty Cycle 3 (Last)			
313.299	RightTopValve	Close			
313.300	LeftBottomValve	Open			
313.300	Controller	End Right Fill Cycle 2 (Last)			
313.301	LeftBottomLamp	On			
313.301	Controller	Start Right Empty Cycle 2 (Last)			
313.302	RightBottomValve	Open			
313.303	RightBottomLamp	On			
313.500	LeftTopSensor	Off			
313.502	RightTopSensor	Off			
318.502	RightMiddleSensor	Off			
321.301	LeftBottomLamp	Off			
321.303	RightBottomLamp	Off			
323.502	RightBottomSensor	Off			
323.503	RightBottomValve	Close			
323.504	Controller	End Right Empty Cycle 2 (Last)			
338.500	LeftMiddleSensor	Off			
363.500	LeftBottomSensor	Off			
363.501	LeftBottomValve	Close			
363.502	Controller	End Left Empty Cycle 3 (Last)			

Eingabe-Parameter und Annahmen über das Systemverhalten		
Variable	Wert	Typ
cycleCountLeftTank	3	Parameter Anzahl linker Zyklen
cycleCountRightTank	2	Parameter Anzahl rechter Zyklen
runCount	2	Parameter Anzahl Abläufe
lampTime	8	Parameter Zeit, wie lange eine Lampe beim Leeren/Füllen brennt
pumpPause	10	Parameter Wartezeit der Pumpe zwischen zwei Pumpvorgängen
initTime	40	Annahme Zeit für den gesamten Init-Vorgang
messageTime	0	Annahme Zeit zum Verschicken einer Meldung von Objekt zu Objekt
tagTimeValveSensor	0.2	Annahme Zeit vom Öffnen eines Ventils bis zur Anzeige des Sensors
availableWater	200	Annahme Gesamte Wassermenge im System (cl) = 2 volle Seitentanks
flowLeftUpperHose	2	Annahme Durchfluss des linken oberen Schlauches (cl/sek)
flowLeftBottomHose	2	Annahme Durchfluss des linken unteren Schlauches (cl/sek)
flowRightUpperHose	10	Annahme Durchfluss des rechten oberen Schlauches (cl/sek)
flowRightBottomHose	10	Annahme Durchfluss des rechten unteren Schlauches (cl/sek)
flowPump	12	Annahme Pumpleistung vom Bottomtank zum TopTank (cl/sek)

Generelle Anmerkungen		
- Da angenommen werden darf, dass der TopTank ein unendliches Reservoir darstellt, wurde der TopTank in den Szenarien nicht modelliert.		
- Aus dem gleichen Grund wurde auch das Pumpverhalten zwischen dem BottomTank und dem TopTank nicht modelliert.		

Illustration 7: Szenario 2

Simulierte Log-File als Resultat des Szenarios			Beschreibung des Szenarios		
Zeit	Objekt	Nachricht			
0.000	Controller	Start Init	Regulärer Ablauf mit sehr kurzer lampTime.		
40.000	Controller	End Init			
40.001	Controller	Start Run 1			
40.002	System	Start Pumping			
56.669	BottomTankSensor	Off			
56.670	System	Stop Pumping			
56.671	Controller	Start Left Fill Cycle 1			
56.672	LeftTopValve	Open			
56.673	LeftTopLamp	On			
56.674	LeftBottomSensor	On			
57.673	LeftTopLamp	Off			
61.872	LeftMiddleSensor	On			
66.872	LeftTopSensor	On			
66.873	LeftTopValve	Close			
66.874	Controller	End Left Fill Cycle 1			
66.875	Controller	Start Left Empty Cycle 1			
66.876	LeftBottomValve	Open			
66.877	LeftBottomLamp	On			
67.076	LeftTopSensor	Off			
67.077	LeftBottomLamp	Off			
72.076	LeftMiddleSensor	Off			
77.076	LeftBottomSensor	Off			
77.077	LeftBottomValve	Close			
77.078	Controller	End Left Empty Cycle 1			
77.079	Controller	Start Left Fill Cycle 2			
77.080	LeftTopValve	Open			
77.081	LeftTopLamp	On			
77.280	LeftBottomSensor	On			
78.081	LeftTopLamp	Off			
82.280	LeftMiddleSensor	On			
87.280	LeftTopSensor	On			
87.281	LeftTopValve	Close			
87.282	Controller	End Left Fill Cycle 2			
87.283	Controller	Start Left Empty Cycle 2			
87.284	LeftBottomValve	Open			
87.285	LeftBottomLamp	On			
87.484	LeftTopSensor	Off			
88.285	LeftBottomLamp	Off			
92.484	LeftMiddleSensor	Off			
97.484	LeftBottomSensor	Off			
97.485	LeftBottomValve	Close			
97.486	Controller	End Left Empty Cycle 2			
97.487	Controller	Start Left Fill Cycle 3 (Last)			
97.488	LeftTopValve	Open			
97.489	LeftTopLamp	On			
97.688	LeftBottomSensor	On			
98.489	LeftTopLamp	Off			
102.688	LeftMiddleSensor	On			
102.689	Controller	Start Right Fill Cycle 1			
102.690	RightTopValve	Open			
102.691	RightTopLamp	On			
102.890	RightBottomSensor	On			
103.691	RightTopLamp	Off			
107.688	LeftTopSensor	On			
107.689	LeftTopValve	Close			
107.690	Controller	End Left Fill Cycle 3 (Last)			
107.890	RightMiddleSensor	On			
112.890	RightTopSensor	On			
112.891	RightTopValve	Close			
112.892	Controller	End Right Fill Cycle 1			
112.893	Controller	Start Right Empty Cycle 1			
112.894	RightBottomValve	Open			
112.895	RightBottomLamp	On			
113.094	RightTopSensor	Off			
113.895	RightBottomLamp	Off			
118.094	RightMiddleSensor	Off			
123.094	RightBottomSensor	Off			
123.095	RightBottomValve	Close			
123.096	Controller	End Right Empty Cycle 1			
123.097	Controller	Start Right Fill Cycle 2 (Last)			
123.098	RightTopValve	Open			
123.099	RightTopLamp	On			
123.298	RightBottomSensor	On			
124.099	RightTopLamp	Off			
128.298	RightMiddleSensor	On			
133.298	RightTopSensor	On			
133.299	Controller	Start Left Empty Cycle 3 (Last)			
133.299	RightTopValve	Close			
133.300	LeftBottomValve	Open			
133.300	Controller	End Right Fill Cycle 2 (Last)			
133.301	LeftBottomLamp	On			
133.301	Controller	Start Right Empty Cycle 2 (Last)			
133.302	RightBottomValve	Open			
133.303	RightBottomLamp	On			
133.500	LeftTopSensor	Off			
133.502	RightTopSensor	Off			
134.301	LeftBottomLamp	Off			
134.303	RightBottomLamp	Off			
138.500	LeftMiddleSensor	Off			
138.502	RightMiddleSensor	Off			
143.500	LeftBottomSensor	Off			
143.501	LeftBottomValve	Close			
143.502	Controller	End Left Empty Cycle 3 (Last)			
143.502	RightBottomSensor	Off			
143.503	RightBottomValve	Close			
143.504	Controller	End Right Empty Cycle 2 (Last)			

Eingabe-Parameter und Annahmen über das Systemverhalten		
Variable	Wert	Typ
cycleCountLeftTank	3	Parameter Anzahl linker Zyklen
cycleCountRightTank	2	Parameter Anzahl rechter Zyklen
runCount	2	Parameter Anzahl Abläufe
lampTime	1	Parameter Zeit, wie lange eine Lampe beim Leeren/Füllen brennt
pumpPause	10	Parameter Wartezeit der Pumpe zwischen zwei Pumpvorgängen
initTime	40	Annahme Zeit für den gesamten Init-Vorgang
messageTime	0	Annahme Zeit zum Verschicken einer Meldung von Objekt zu Objekt
tagTimeValveSensor	0.2	Annahme Zeit vom Öffnen eines Ventils bis zur Anzeige des Sensors
availableWater	200	Annahme Gesamte Wassermenge im System (cl) = 2 volle Seitentanks
flowLeftUpperHose	10	Annahme Durchfluss des linken oberen Schlauches (cl/sek)
flowLeftBottomHose	10	Annahme Durchfluss des linken unteren Schlauches (cl/sek)
flowRightUpperHose	10	Annahme Durchfluss des rechten oberen Schlauches (cl/sek)
flowRightBottomHose	10	Annahme Durchfluss des rechten unteren Schlauches (cl/sek)
flowPump	12	Annahme Pumpleistung vom Bottomtank zum TopTank (cl/sek)

Generelle Anmerkungen		
- Da angenommen werden darf, dass der TopTank ein unendliches Reservoir darstellt, wurde der TopTank in den Szenarien nicht modelliert.		
- Aus dem gleichen Grund wurde auch das Pumpverhalten zwischen dem BottomTank und dem TopTank nicht modelliert.		

Illustration 8: Szenario 3

## **9. OO-Synthese**

### **9.1. Vorgehen nach Godet**

Teil der Objektorientierten Synthese nach Godet ist das Sammeln der Adjektive und Verben im bisher erarbeiteten Text, um auf das Fachklassendiagramm zu kommen. Dabei werden die verschiedenen Wortarten in verschiedenen Farben eingefärbt.

### **9.2. Einfärben der Begriffe**

Wir haben die OO-Synthese anhand des Pflichtenheftes und den Use Cases gemacht. Aus gelben Substantiven werden Klassen, aus orangen Substantiven Attribute, aus den lila Verben die Methoden, aus den grünen Zahlwörtern die Multiplizitäten sowie aus den blauen Bedingungen die Bedingungen. Die Unterscheidung zwischen Klassen und Attributen erfolgt schrittweise und nach Ermessen.

Gelb	Substantive	Klassen
Orange	Substantive	Attribute
Lila	Verben	Methoden
Grün	Zahlwörter	Multiplizitäten
Blau	Bedingungen	Bedingungen

## Steuerung "Simple Tank System"

### A. Grobpflichtenheft

Implementation einer Steuerung für die programmgesteuerte Füllung und Leerung von Gefässen, die mittels Ventilen und einer Pumpe miteinander verbunden sind. Die Ablaufparameter sind vom Benutzer wählbar.

Projektphasen: Analyse, Grob-Design, Detail-Design, Programmierung, Test und Einführung (dargestellt durch Projektpräsentation).

### B. Aufgaben

#### 1. Grundlagen

- System-Architektur SITAS: Beilage 1,
- Simple Tank System (Hardware-Disposition und Steuerelemente): Beilage 2,
- Interface-Beschreibung (Java-Doc): <https://prof.ti.bfh.ch/index.php?id=1941>,
- Objekt-Diagramm von sitas.client: Beilage 3,
- Beispiel-Programm: Beilage 4,
- Anleitung zur Benützung des realen Modells: Beilage 5,
- Anleitung zur Projekteröffnung in Eclipse: Beilage 6

#### 2. Benutzerinteraktion

Der Benutzer kann am Bildschirm die Anzahl Füllen-/Leeren-Zyklen je für den linken und rechten Tank und die Anzahl Abläufe (unter einem Ablauf sei die Sequenz der Zyklen des linken und des rechten Tanks verstanden) sowie weitere Steuerparameter angeben.

#### 3. Steueralgorithmus

- a. In einer Initialisierungsphase ist der obere Tank zu füllen und der linke und rechte Tank zu leeren.
- b. Die Steuerparameter nCycleCount\_leftTank, nCycleCount\_rightTank, nRunCount, nLampTime und nPumpPause sind einzulesen.
- c. Steuer-Algorithmus:
  1. Der obere Tank stellt ein "unendliches" Reservoir dar, d.h. der obere Tank ist zu füllen, bis der untere Tank keine Flüssigkeit mehr hat, danach ist die Pumpe für nPumpPause Sekunden auszuschalten (Schonung der Pumpe),
  2. Der linke Tank wird nCycleCount\_leftTank-mal gefüllt und geleert; die letzte Leerung wird aber erst initiiert, wenn der rechte Tank in seinem letzten Zyklus voll geworden ist,
  3. der Zyklus des rechten Tanks wird gestartet, sobald der linke Tank im letzten Zyklus halbvoll geworden ist; der rechte Tank wird dann nCycleCount\_rightTank-mal gefüllt und geleert,
  4. sobald der rechte Tank in seinem letzten Zyklus voll geworden ist, wird das Ende des linken Tank-Zyklus gestartet,
  5. Beim Füllen und Leeren sind die entsprechenden Lampen für nLampTime Sek. anzuzünden (Minimalforderung, es kann aber auch ein besseres Anzeigeschema implementiert werden),
  6. Ein Ablauf endet mit dem Leerwerden des letzten der beiden Tanks (linker resp. rechter Tank),
  7. Die Anzahl der Abläufe bestimmt, wie oft der oben beschriebene Ablauf laufen soll.
  8. Wird Anzahl der Abläufe = 0 eingegeben, so soll das Programm beendet werden.
- d. Ein Logging-System soll mittels Textfile und einer einfachen SQL-Datenbank realisiert werden; der wahlweise Betrieb des Loggings soll möglich sein.
- e. Mittels "Reset" am Modell kann der obige Algorithmus jederzeit unterbrochen werden und das System soll eine Fehlermeldung anzeigen.

### Illustration 9: OO-Synthese mittels Sammeln von Begriffen

Aus den gesammelten Begriffen wird anschliessend eine Tabelle der Fachklassen, deren Attribute und Methoden, sowie der Multiplizitäten und Bedingungen erstellt. Unser Beispiel eines solchen Vorgehens finden Sie unten.

### 9.3. Erstellen einer Tabelle der Designelemente

Fachklassen	Attribute	Methoden
Gesamtsystem		
Bottom Tank	Füllstand	- füllen - leeren
Left Tank	Füllstand	
Right Tank	Füllstand	
Top Tank		
Ventile	Zustand	- öffnen - schliessen
Lampen	Zustand	- anzünden - auslöschen
Pumpen	Zustand	- einschalten - ausschalten
Resettaste		Algorithmus unterbrechen
Logdatei		- öffnen - schliessen - Eintrag schreiben - Einträge darstellen
LogDB		- öffnen - schliessen - Eintrag schreiben - Einträge darstellen
Logeintrag		- erstellen - eintragen
Steuerparameter	nCycleCount_leftTank nCycleCount_rightTank nRunCount nLampTime nPumpPause	- einlesen - überprüfen
Zyklus	Anzahl	- starten - abbrechen
Ablauf	Anzahl	- starten - abbrechen
Initialisierungsphase		- starten - abbrechen
Steueralgorithmus		
Flüssigkeit		
Leerung	Anzahl	
Loggingsystem		

Multipizitäten	Bedingungen / Periodizitäten
4 Tanks	Pumpe läuft nur, wenn im Bottom Tank Wasser vorhanden ist
1 Pumpe	Pumpe darf nur laufen, wenn sie für nPumpPause ausgeschaltet war
5 Ventile	Die letzte Leerung wird aber erst initiiert, wenn der rechte Tank in seinem letzten Zyklus voll geworden ist
4 Lampen	Der Zyklus des rechten Tanks wird gestartet, sobald der linke Tank im letzten Zyklus halbvoll geworden ist
Beliebig viele Logeinträge	Sobald der rechte Tank in seinem letzten Zyklus voll geworden ist, wird das Ende des linken Tank-Zyklus gestartet
Beliebig viele Abläufe	Ein Ablauf endet mit dem Leerwerden des letzten der beiden Tanks (linker resp. rechter Tank),

<b>Assoziationen</b>
Tank – Sensoren
Ventile – Tank (Füllen/Leeren)
Gesamtsystem – Tank
Gesamtsystem – Pumpe
Gesamtsystem – Ventile
Gesamtsystem – Resettaste
Gesamtsystem – Lampen
Left/Right Tank – Lampen

# 10. Prototypen

## 10.1. Prototyp für Initialisierung

Für unsere musikbegleitete Initialisierung haben wir einen Prototypen erstellt. Mit ihm konnten wir auch gleich mal die Ansteuerung des SITAS testen. Die Synchronisierung zu der Musik funktioniert mittels fest programmierten Delays.

Hier der Code für die Delays:

```
/** Times */
private final int BLINKTIME = 100;           // Duration of a single blink
private final int FULLTIME = 2100;            // Duration of the full flash
private final int SHORT_PAUSE = 290;          // Duration of a short pause
private final int LONG_PAUSE = 1200;           // Duration of a long pause
private final int LEFT_RIGHT = 538;           // Pause between left/right blinks
```

Hier der Code für den Start des Files:

```
public void run() {
    System.out.println("Show-Off Init Sequence Starts:");
    System.out.flush();
    tankSystem.reset();

    // Start the sound and wait for the drum section to pass
    playSound();
    sleep(9450);

    // Open all valves and start first sequence
    bottomTank.openValve(V.LOWER);
    leftTank.openValve(V.UPPER);
    leftTank.openValve(V.LOWER);
    rightTank.openValve(V.UPPER);
    rightTank.openValve(V.LOWER);
    playSequence();

    // Turn all on lamps and start/stop pump for the intermezzo
    leftTank.closeValve(V.UPPER);
    rightTank.closeValve(V.UPPER);
    tankSystem.startPump();
    allLamps(FULLTIME);
    sleep(SHORT_PAUSE);
    tankSystem.stopPump();
    leftTank.openValve(V.UPPER);
    rightTank.openValve(V.UPPER);

    ...
}

/**
 * Plays a lamp sequence according to "The eye of the tiger"
 */
private void playSequence() {
    blink(Lamp.LEFT_UPPER);
    sleep(LONG_PAUSE);
    blink(Lamp.LEFT_LOWER);
    sleep(SHORT_PAUSE);
    blink(Lamp.RIGHT_LOWER);
    sleep(SHORT_PAUSE);
    blink(Lamp.RIGHT_UPPER);
    sleep(LONG_PAUSE);
    blink(Lamp.LEFT_UPPER);
    sleep(SHORT_PAUSE);
    blink(Lamp.LEFT_LOWER);
    sleep(SHORT_PAUSE);
    blink(Lamp.RIGHT_LOWER);
    sleep(LONG_PAUSE);
    blink(Lamp.RIGHT_UPPER);
    sleep(SHORT_PAUSE);
    blink(Lamp.LEFT_UPPER);
    sleep(SHORT_PAUSE);
}
```

## 10.2. UI-Prototyp

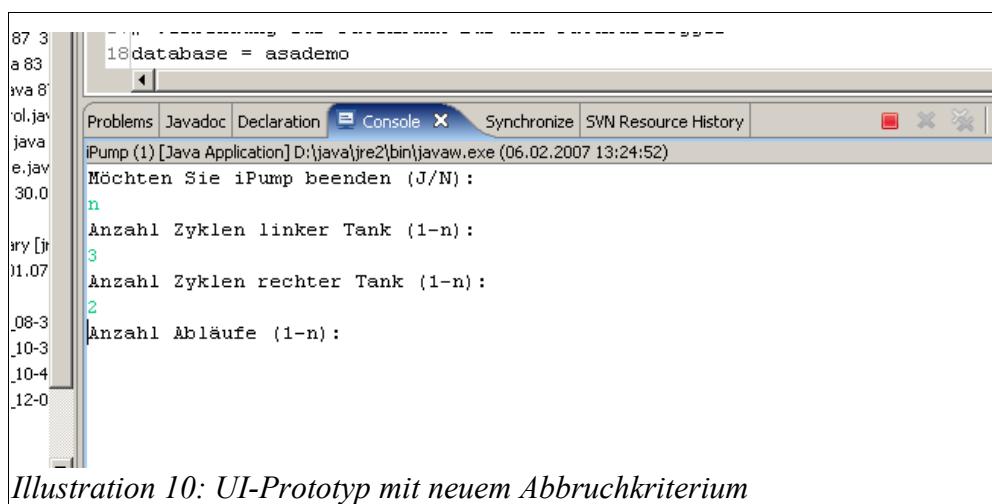
Ziel des UI-Prototyps war es, die Darstellung der Eingaben zu prüfen, sowie mit dem Auftraggeber abzuklären, ob die ursprünglich verlangte Eingabe von '0' als Abbruchkriterium durch eine Ja/Nein-Frage ersetzt werden kann. Dies ist kein Mehraufwand und das Interface profitiert durch eine klarere Aufmachung.

Codebeispiel: Ja/Nein-Frage zum Programmabbruch

```
public Parameter getParameter(Parameter parameter) {
    Scanner in;

    // End program?
    in = new Scanner(System.in);
    printLine("Möchten Sie iPump beenden (J/N):");
    if (in.hasNext()) {
        String endProgram = in.next();
        if (endProgram.equalsIgnoreCase("J")) {
            setEndOfProgram(true);
            return null;
        }
    }
}
```

Aussehen im UI:



# 11. Fachklassen-Diagramm und Beschreibung

## 11.1. Fachklassendiagramm

Unser Fachklassendiagramm entstand auf der Grundlage der Objektorientierten Synthese und einer Diskussion in der Gruppe. Verschiedene Fragen waren noch offen, unter anderem die Verteilung der Parameter-Daten und das Logging. Unten finden Sie ein Diagramm, wie es in der frühen Phase aussah.

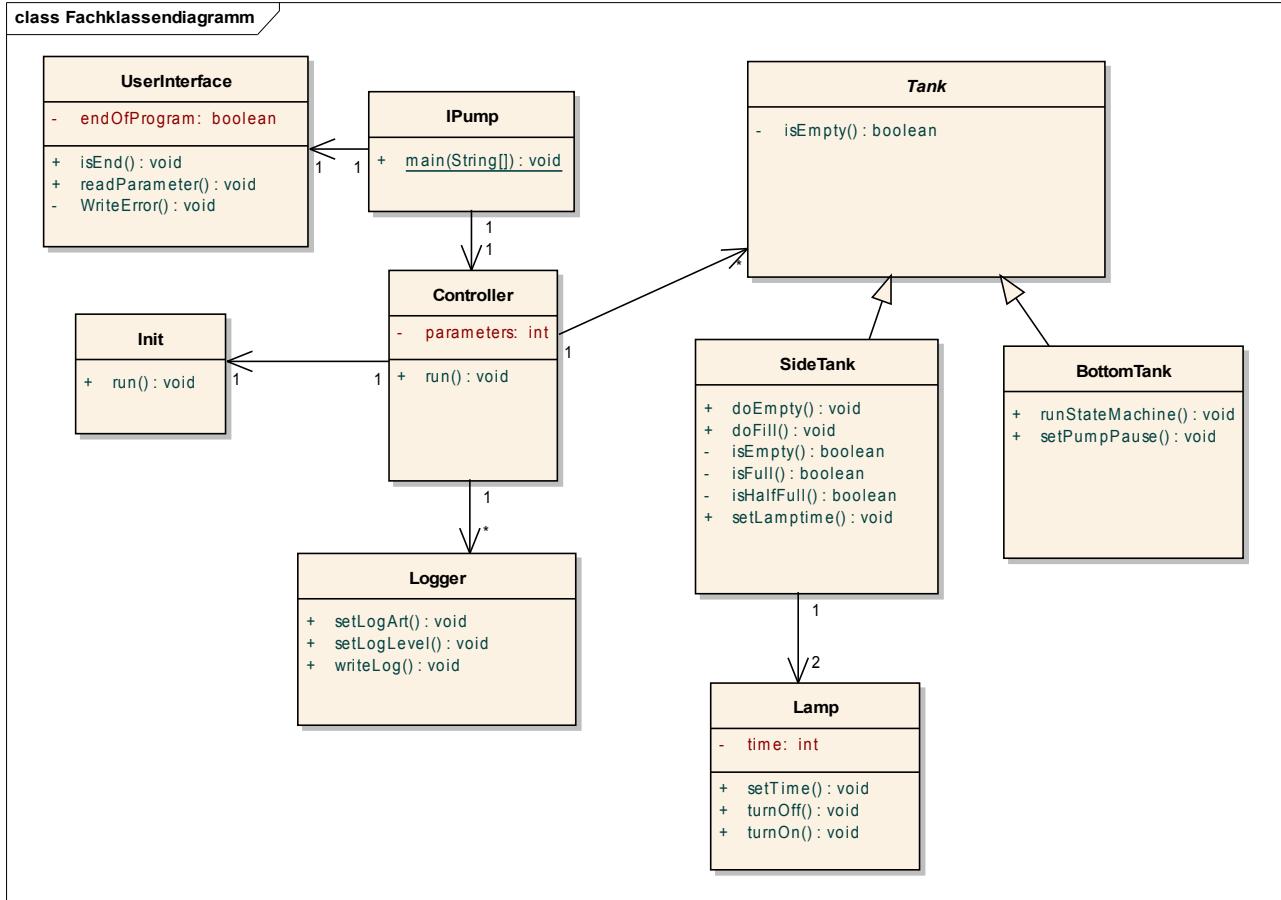


Illustration 11: Fachklassendiagramm

## 11.2. Beschreibung

Die Klassen haben folgende Aufgaben (Auszug aus dem EAP-File):

### BottomTank

**public Class**

**Extends:** *Tank*. : Der Bottom tank mit der Pumpe stellt sicher, dass immer genügend Wasser im Toptank ist, aber gleichzeitig die Pausetime eingehalten wird.

### Controller

**public Class:** Der Controller ist der Kern der Applikation. Er verwaltet die Tanks und steuert deren Füllen und Leeren. Er beinhaltet eine State-Event-Machine die die Abläufe sicherstellt.

### Init

**public Class:** Init ist die Initialisierung. Nach dem Ablauf von Init steht das System im Ausgangszustand für die Zyklen bereit, der obere Tank ist gefüllt.

### IPump

**public Class:** Hauptfunktion, die von der Konsole aufgerufen wird.

### Lamp

**public Class:** Eine Lampe kann ein -und ausgeschaltet werden, sowie nach einer bestimmten Zeit sich selbst löschen. Dies soll selbstständig mittels eines Timers gelöst werden.

### Logger

**public Class:** Der Logger loggt alle relevanten Events. Für Debugging stehen verschiedene Loglevels zur Verfügung. Er handelt die Anbindung an die DB und/oder Logfile selbstständig.

### SideTank

**public Class**

**Extends:** *Tank*. : Side Tank stellt die konkrete Implementation eines linken oder rechten Seitentanks dar. Er besitzt eine StateEventMachine, die genügend komplext ist, damit er füllen und leeren kann, sowie den aktuellen Zustand zurückgibt. Ein Side Tank kann auch die ihm zugehörigen Lampen selbstständig bedienen.

### Tank

**public abstract Class:** Tank ist eine abstrakte Klasse für Tanks.

### UserInterface

**public Class:** Das User Interface stellt die Kommunikation mit dem Benutzer sicher. Via Benutzereingabe muss das Programm abgebrochen werden können. Dies wird mit isEnd() geprüft.

# 12. Detaildesign der Sensoren mit Fehlerkorrektur

## 12.1. Logik

Die Fehlerkorrektur wurde statisch gelöst, in Absprache mit dem Auftraggeber. Das heisst, die Vergangenheit wird nicht berücksichtigt.

Die Fehlerkorrektur ist in den Methoden:

- isEmpty()
- isHalfFull()
- isFull()

implementiert. Diese Methoden entscheiden anhand der Sensoren und des Tankzustandes (d.h. ob er am füllen oder am leeren ist) ob der Tank voll, halbvoll oder leer ist.

Resultat der Methoden: Funktionen geben TRUE wenn:	Zustand der Sensoren: (hasSensorLiquid())			Zustand Tank: FILLING / !EMPTYING
	EMPTY	HALF	FULL	
1 isEmpty()	○	○	○	○
2 isEmpty()	○	○	○	●
3 isEmpty()	○	○	●	○
4 isEmpty()	○	○	●	●
5 isEmpty()	○	●	○	○
6 isHalfFull()	○	●	○	●
7 isFull()	○	●	●	○
8 isFull()	○	●	●	●
9 isHalfFull()	●	○	○	○
10 isEmpty()	●	○	○	●
11 isFull()	●	○	●	○
12 isFull()	●	○	●	●
13 isFull()	●	●	○	○
14 isHalfFull()	●	●	○	●
15 isFull()	●	●	●	○
16 isFull()	●	●	●	●

Vereinfacht ergibt sich daraus folgende Logik:

```
isEmpty =
(!EMPTY && !HALF) ||
(FILLING && !FULL && !HALF) ||
(!FILLING && !FULL && !EMPTY)

isHalfFull =
(FILLING && HALF && !FULL) ||
(!FILLING && EMPTY && !HALF && !FULL)

isFull =
(EMPTY & FULL) ||
(HALF & FULL) ||
(!FILLING && EMPTY && HALF && !FULL)
```

## 12.2. Whiteboxtest (Codebeispiel)

```
public class UpperTest {

    static boolean sensEmpty = false;
    static boolean sensHalf = false;
    static boolean sensFull = false;
    static boolean filling = false;

    public static void main(String[] args) {
        setValues(false, false, false, false);
        setValues(false, false, false, true);
        setValues(false, false, true, false);
        setValues(false, false, true, true);
        setValues(false, true, false, false);
        setValues(false, true, false, true);
        setValues(false, true, true, false);
        setValues(false, true, true, true);
        setValues(true, false, false, false);
        setValues(true, false, false, true);
        setValues(true, false, true, false);
        setValues(true, false, true, true);
        setValues(true, true, false, false);
        setValues(true, true, false, true);
        setValues(true, true, true, false);
        setValues(true, true, true, true);
    }

    public static void setValues(boolean E, boolean H, boolean F, boolean S){
        sensEmpty = E;
        sensHalf = H;
        sensFull = F;
        filling = S;

        System.out.print("isEmpty(): " + isEmpty());
        System.out.print("  isFull(): " + isFull());
        System.out.println("  isHalfFull(): " + isHalfFull());
    }

    public static boolean isEmpty(){
        return (
            (!sensEmpty && !sensHalf) ||
            (filling && !sensFull && !sensHalf) ||
            (!filling && !sensFull && !sensEmpty)
        );
    }

    public static boolean isFull(){
        return (
            (sensEmpty & sensFull) ||
            (sensHalf & sensFull) ||
            (!filling && sensEmpty && sensHalf && !sensFull)
        );
    }

    public static boolean isHalfFull(){
        return (
            (filling && sensHalf && !sensFull) ||
            (!filling && sensEmpty && !sensHalf && !sensFull)
        );
    }
}
```

## 12.3. Whiteboxtest (Consolen Printout)

```
1 isEmpty(): true   isFull(): false    isHalfFull(): false
2 isEmpty(): true   isFull(): false    isHalfFull(): false
3 isEmpty(): true   isFull(): false    isHalfFull(): false
4 isEmpty(): true   isFull(): false    isHalfFull(): false
5 isEmpty(): true   isFull(): false    isHalfFull(): false
6 isEmpty(): false  isFull(): false    isHalfFull(): true
7 isEmpty(): false  isFull(): true     isHalfFull(): false
8 isEmpty(): false  isFull(): true     isHalfFull(): false
9 isEmpty(): false  isFull(): false    isHalfFull(): true
10 isEmpty(): true  isFull(): false   isHalfFull(): false
11 isEmpty(): false isFull(): true    isHalfFull(): false
12 isEmpty(): false isFull(): true    isHalfFull(): false
13 isEmpty(): false isFull(): true    isHalfFull(): false
14 isEmpty(): false isFull(): false   isHalfFull(): true
15 isEmpty(): false isFull(): true    isHalfFull(): false
16 isEmpty(): false isFull(): true    isHalfFull(): false
```

# 13. Software-Design

Wir haben den kompletten Design mit dem CASE-Tool Enterprise Architect erstellt. Hier haben wir die für die Dokumentation verlangten Diagramme und Beschreibungen eingefügt. Eine vollständige und verlinkte Version dieser Dokumente finden Sie auf der Dokumentations-CD.

## 13.1. Systemklassendiagramm

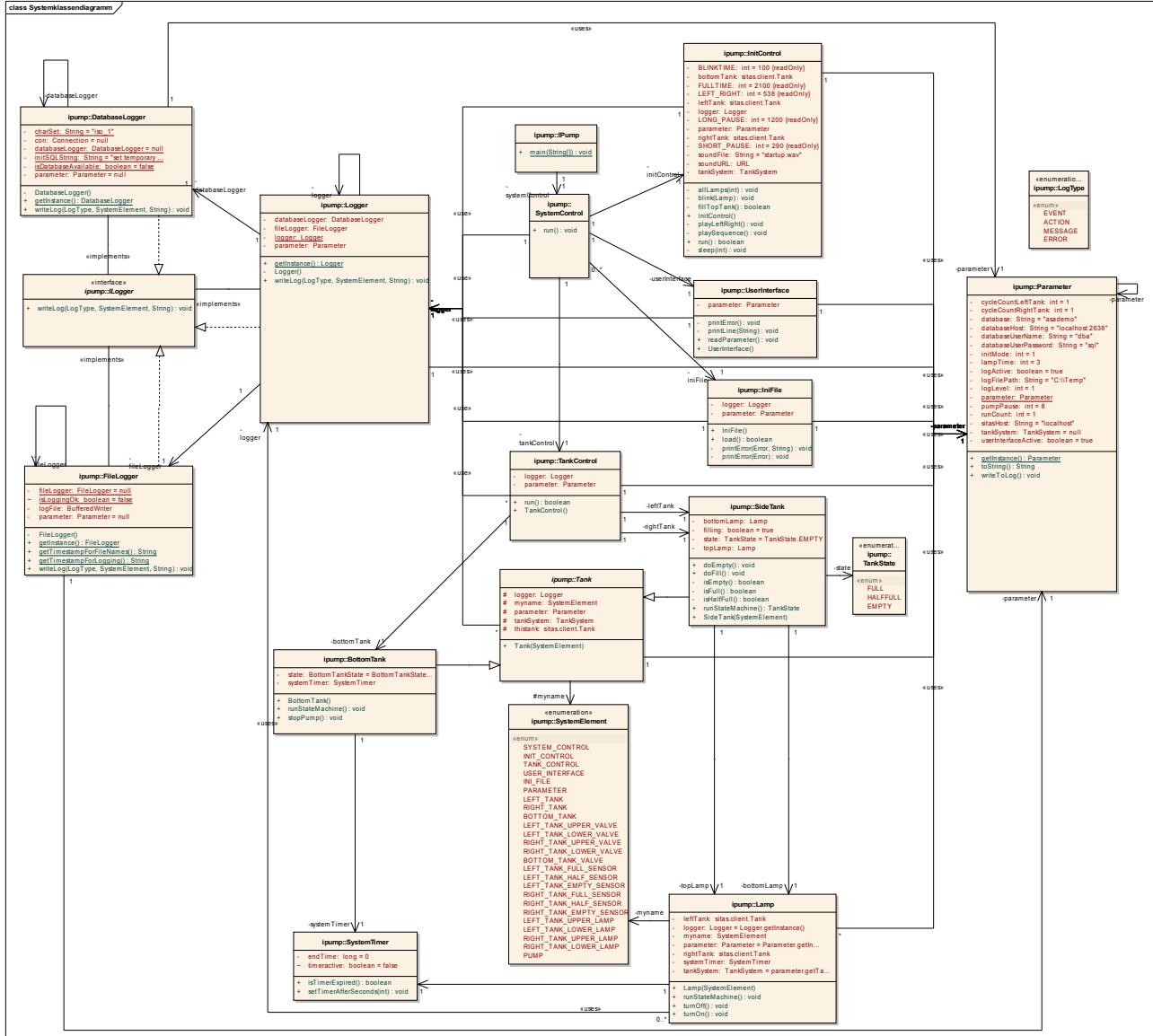


Illustration 12: Systemklassendiagramm

## 13.2. Zustandsdiagramme

### 13.2.1 Klasse BottomTank

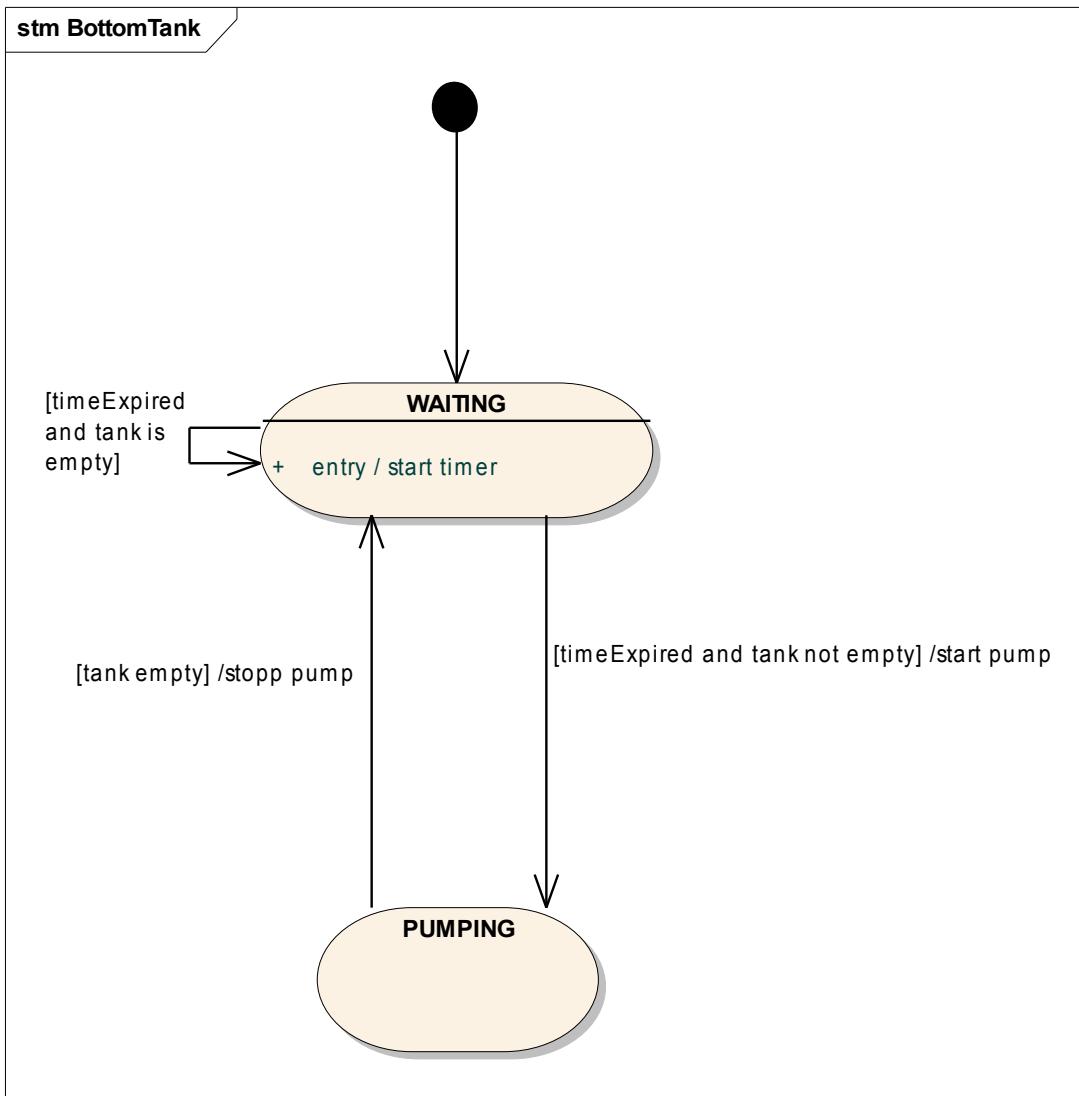


Illustration 13: Zustandsdiagramm für `BottomTank`

### 13.2.2 Klasse Lamp

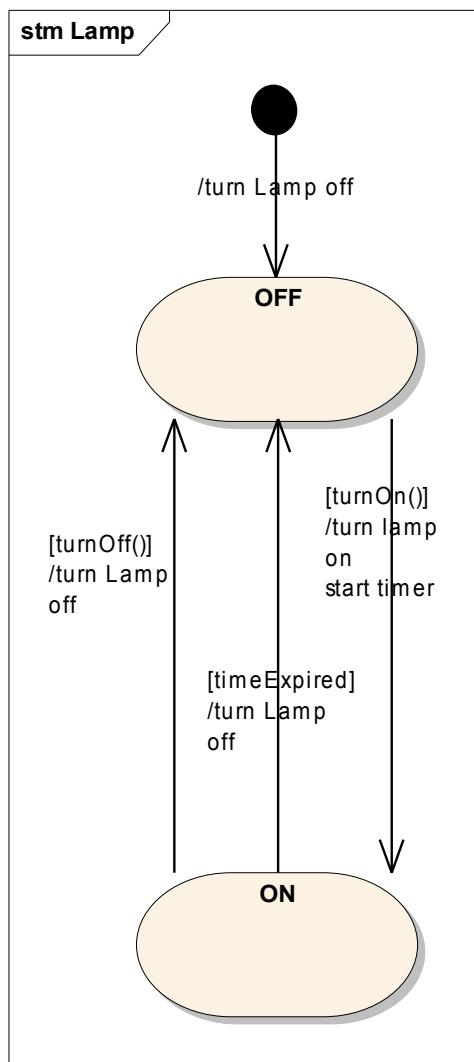


Illustration 14: Zustandsdiagramm  
für Lamp

### 13.2.3 Klasse SideTank

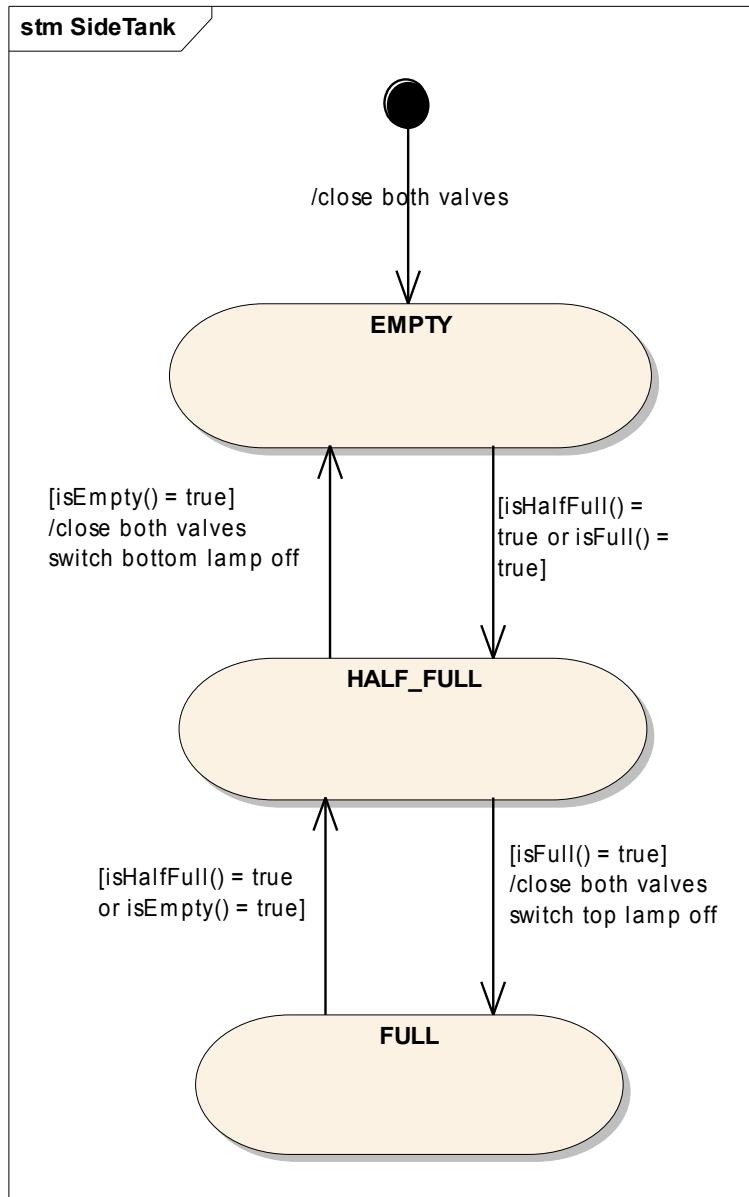


Illustration 15: Zustandsdiagramm für SideTank

Die Seitentanks verfügen über drei Zustände, die durch die drei Sensoren an den Tanks gegeben sind. Eine Zustandsänderung von zwischen Leer und Voll ist bewusst nicht möglich, es führt immer über den Zustand "halb Voll". Dieser Mechanismus ist so gewählt, um Fehler im Programmablauf zu vermieden, falls der mittlere Sensor nicht angeben würde. Da der rechte Tank auf das Ereignis "halb Voll" des linken angewiesen ist, darf dieser Zustand nicht übersprungen werden.

### 13.2.4 Klasse TankControl

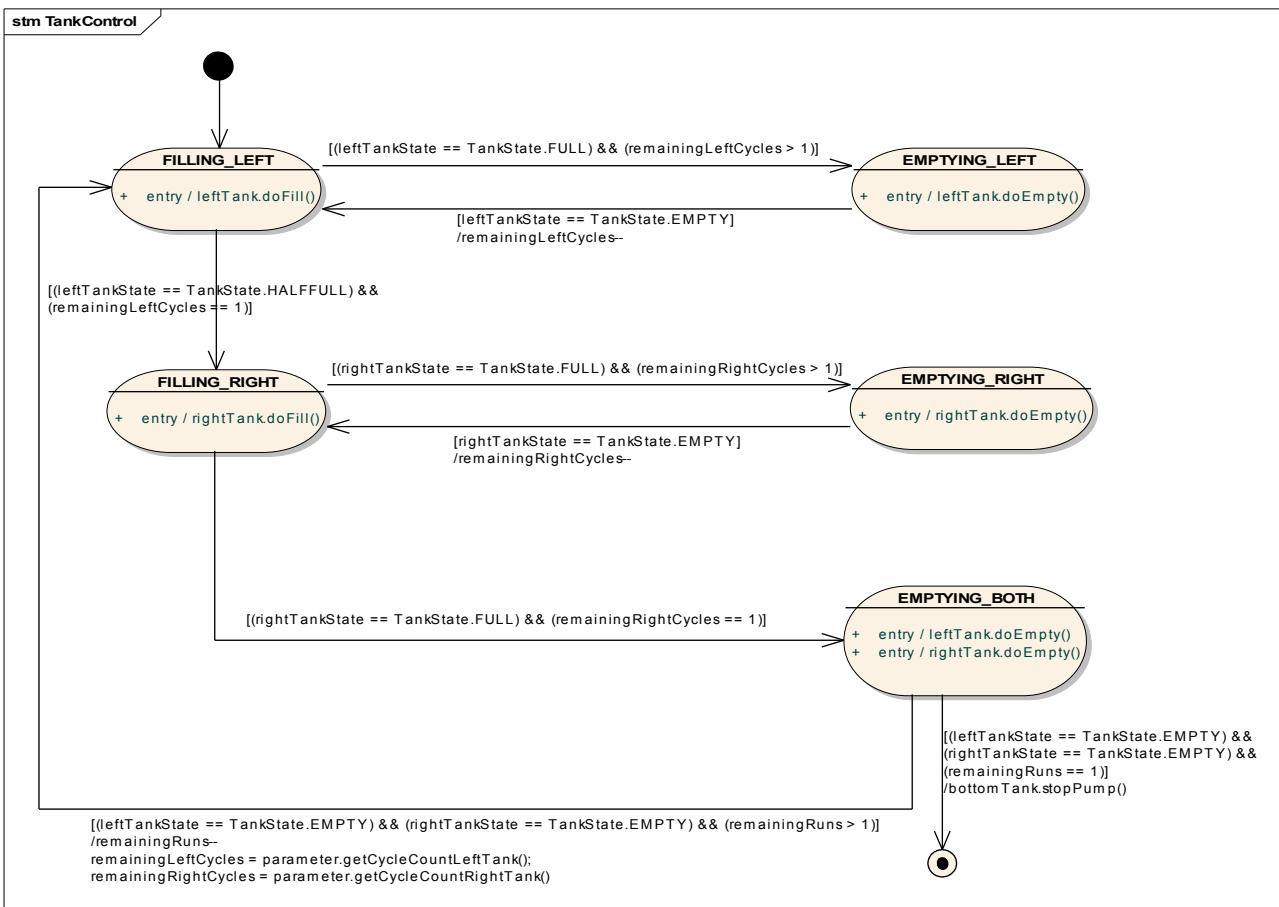


Illustration 16: Zustandsdiagramm für TankControl

### 13.3. Sequenzdiagramm

Wir haben ein Sequenzdiagramm für den Startup-Vorgang inkl. eines vereinfachten Ablaufes der State-Event Maschine über alle Klassen erstellt.

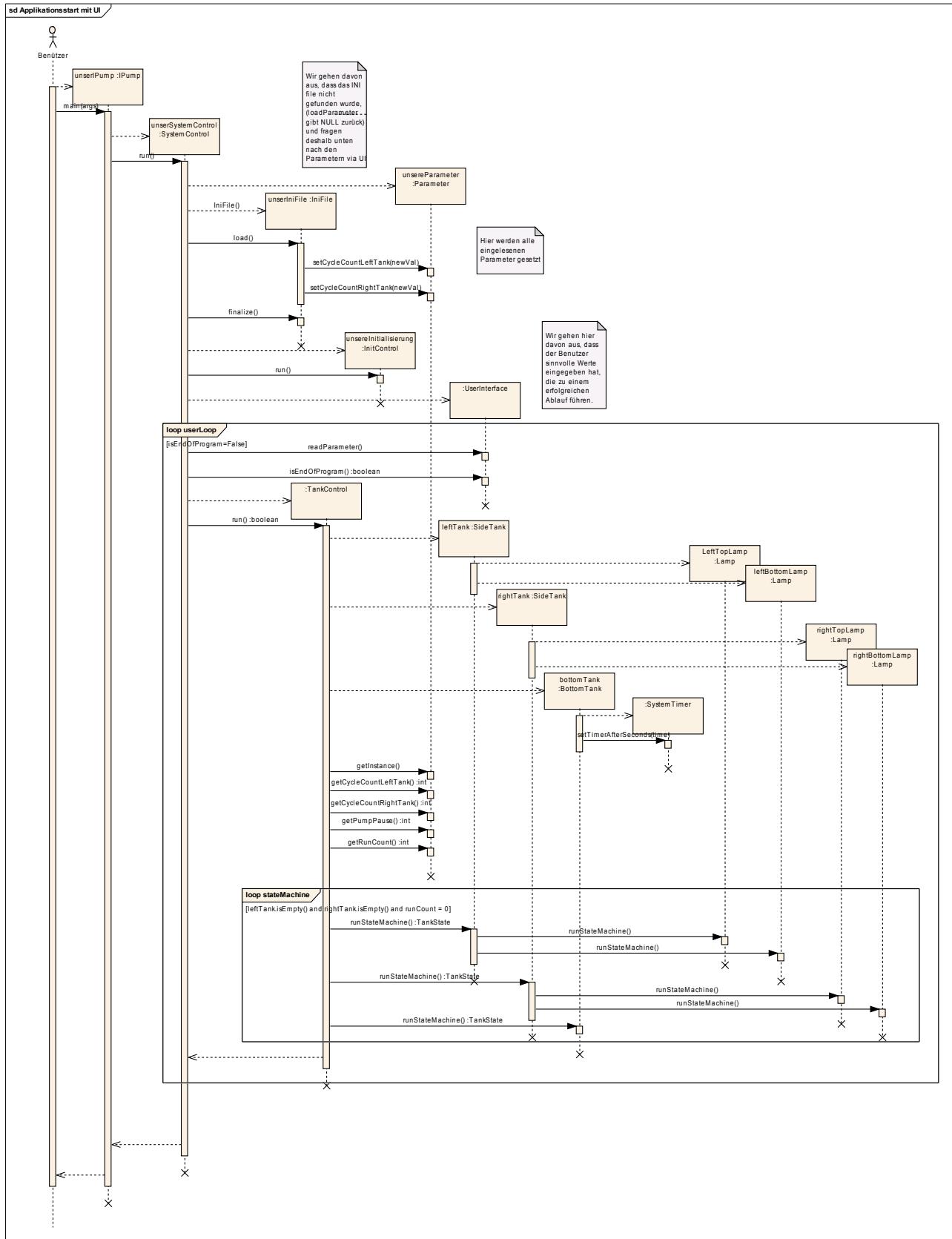


Illustration 17: Sequenzdiagramm für Applikationsstart

## 13.4. Attribut- und Operationenbeschreibung

Die nachfolgende Attribut- und Operationenbeschreibung ist ein RTF-Export aus dem EAP. Für eine Übersichtliche Darstellung verweisen wir auch hier auf die Dokumentations-CD, welche die komplette Dokumentation im HTML-Format enthält.

### ipump::BottomTank

public Class

**Extends:** *Tank*. : Klasse für den BottomTank. Steuert selbständig den Bottomtank. Dazu gehört der Tank, die Pumpe und das Ventil. Steuert auch die Wartezeit der Pumpe. Die Statemachine muss periodisch aufgerufen werden. Die Statemachine läuft unendlich. Vor den Zerstören des Objektes muss die Pumpe von aussen durch die Funktion stopPump beendet werden.

#### ipump::BottomTank Attributes

Attribute	Type	Notes
state	private : <i>BottomTankState</i>	Interner Zustand des BottomTanks Initial Value: <i>BottomTankState.WAITING</i> ;
systemTimer	private : <i>SystemTimer</i>	Referenz auf ein SystemTimer Objekt

#### ipump::BottomTank Methods

Method	Type	Notes
BottomTank ()	public:	Konstruktor stoppt Pumpe schliesst Ventil startet Timer
runStateMachine ()	public: void	StateMachine des Bottomtanks wird von der Tankcontrol im Systemtank aufgerufen. Ist eine Endlos Statemachine welche zwischen Pumpen und Warten wechselt.
stopPump ()	public: void	Stoppt die Pumpe und schliesst das Ventil zwischen TopTank und BottomTank.

### ipump::BottomTank::BottomTankState

package «enumeration» Class: Interner Zustand des BottomTanks

#### ipump::BottomTank::BottomTankState Attributes

Attribute	Type	Notes
PUMPING «enum»	public :	
WAITING «enum»	public :	

### PUMPING

public State:

### WAITING

public State:

#### WAITING Methods

Method	Type	Notes
start timer ()	public: entry	

## <anonymous>

*public Initial State:*

## ipump::DatabaseLogger

*public Class*

**Implements: ILogger.** : Übernimmt die Kommunikation zur Datenbank und schreibt die erhaltenen Log-Einträge in die bereits vorhandene Datenbanktabelle.

### ipump::DatabaseLogger Attributes

Attribute	Type	Notes
databaseLogger	private static : DatabaseLogger	Instanz des DatabaseLogger, gebraucht für Singleton Pattern Initial Value: null;
parameter	private : Parameter	Referenz auf Parameter Initial Value: null;
con	private : Connection	DatabaseConnection Initial Value: null;
charSet	private static : String	Charakter Set für Datenbank Initial Value: "iso_1";
initSQLString	private static : String	Initialisierungsstrings für die Datenbank Initial Value: "set temporary option date_order = 'DMY';" + "set temporary option return_date_time_as_string = 'on';" + "set temporary option date_format = 'DD-MMM-YYYY';";
isDatabaseAvailable	private static : boolean	Flag zur Anzeig ob das DatenbankLogging funktioniert, im Fehlerfall wird das Flag gelöscht, somit wird das DatenbankLogging deaktiviert. Initial Value: false;

### ipump::DatabaseLogger Methods

Method	Type	Notes
DatabaseLogger ()	private:	Stellt eine Verbindung zur Datenbank her, die Verbindungsparameter werden aus dem ini-File gelesen. Kann keine Verbindung zur Datenbank hergestellt werden wird das Database-Logging deaktiviert. Der Benutzer erhält eine Konsolenmeldung, sofern das Database Logging nicht möglich ist.
getInstance ()	public static: DatabaseLogger	Erstellt eine neue Instanz vom DatabaseLogger, ist bereits eine vorhanden, wird die Referenz auf diesen zurückgegeben @return Referenz auf einmalige Instanz des DatabaseLogger
writeLog (LogType, SystemElement, String)	public: void	param: type [ LogType - in ] enthält den Nachrichtentyp param: source [ SystemElement - in ] gibt die Quelle der Nachricht an param: text [ String - in ] enthält die Nachricht  Schreibt einen Log-Eintrag in die Datenbank, sofern das Logging im ini-File aktiviert wurde. Kann nicht mehr auf die Datenbank zugegriffen werden, wird eine einmalige Meldung in die Konsole ausgegeben und das DatabaseLogging deaktiviert.

## ipump::FileLogger

*public Class*

**Implements: ILogger.** : Stellt die nötigen Methoden zum File Logging zur Verfügung und gibt entsprechende Fehlermeldungen in die Konsole aus.

### ipump::FileLogger Attributes

Attribute	Type	Notes
parameter	private : <i>Parameter</i>	Referenz auf Parameter Initial Value: null;
fileLogger	private static : <i>FileLogger</i>	Instanz des FileLogger, gebraucht für Singleton Pattern Initial Value: null;
isLoggingOk	package static : boolean	Flag zur Anzeig ob das FileLogging funktioniert, im Fehlerfall wird das Flag gelöscht, und somit das FileLogging deaktiviert. Initial Value: false;
logFile	private : <i>BufferedWriter</i>	Referenz auf das LogFile

#### *ipump::FileLogger Methods*

Method	Type	Notes
FileLogger ()	private:	Erstellt ein neues Log-File in den gewünschten Ordner, der im ini-File angegeben werden kann. Jeder neue Programmaufruf erstellet ein neues Log-File. Kann das File nicht erstellt werden wird eine entsprechende Meldung in der Konsole ausgegeben.
getInstance ()	public static: <i>FileLogger</i>	Erstellt eine neue Instanz vom FileLogger, ist bereits eine vorhanden, wird die Referenz auf diesen zurückgegeben @return Referenz auf einmalige Instanz des FileLogger
getTimestampForFileNames ()	public static: <i>String</i>	Liefert einen formatierten Zeitstempel der Systemzeit @return gibt einen formatierten Zeitstempel (?) als String zurück
getTimestampForLogging ()	public static: <i>String</i>	Liefert einen formatierten Zeitstempel für die File-Logeinträge der Systemzeit @return gibt einen formatierten Zeitstempel (?) als String zurück
writeLog ( <i>LogType</i> , <i>SystemElement</i> , <i>String</i> )	public: void	param: type [ <i>LogType</i> - in ] enthält den Nachrichtentyp param: source [ <i>SystemElement</i> - in ] gibt die Quelle der Nachricht an param: text [ <i>String</i> - in ] enthält die Nachricht  Schreibt einen Log-Eintrag ins Log-File, sofern das Loggin im ini-File aktiviert wurde. Kann nicht mehr ins File geschrieben werden, wird eine einmalige Meldung in die Konsole ausgegeben und das FileLogging deaktiviert.

#### *ipump::ILogger*

*public abstract «interface» Class:*

#### *ipump::ILogger Methods*

Method	Type	Notes
writeLog ( <i>LogType</i> , <i>SystemElement</i> , <i>String</i> )	public: void	param: type [ <i>LogType</i> - in ] enthält den Nachrichtentyp param: source [ <i>SystemElement</i> - in ] gibt die Quelle der Nachricht an param: text [ <i>String</i> - in ] enthält die Nachricht  Schreibt einen Eintrag ins Log.

#### *ipump::IniFile*

*public Class:* Diese Klasse stellt eine INI-Datei für iPump dar. Sie kann die Datei "ipump/ipump.ini" geladen werden. Ist die INI-Datei nicht vorhanden oder tritt ein Fehler auf, wird <code>null</code> zurückgegeben. Wenn die Datei gelesen werden kann, werden die Daten in das Paramter-Singleton geschrieben.

### *ipump::IniFile Attributes*

Attribute	Type	Notes
logger	private : <i>Logger</i>	Referenz auf das Singleton Logger
parameter	private : <i>Parameter</i>	Referenz auf das Singleton Parameter

### *ipump::IniFile Methods*

Method	Type	Notes
IniFile ()	public:	Konstruktor. Erstellt Referenzen auf die Singletons Logger und Parameter.
load ()	public: boolean	Lädt die INI-Datei und schreibt die gefundenen Werte in das Parameter-Singleton @return boolean Gibt <code>true</code> zurück, wenn die Datei erfolgreich geladen und verarbeitet werden konnte, <code>false</code> in allen anderen Fällen.
printError (Error, String)	private: void	param: error [ Error - in ] Aufgetretener Fehler param: nameOfProperty [ String - in ] Name des Wertes, bei dem der Fehler aufgetreten ist  Schreibt eine Fehlermeldung auf die Konsole und ins Log (über den Logger).
printError (Error)	private: void	param: error [ Error - in ] Aufgetretener Fehler  Schreibt eine Fehlermeldung auf die Konsole und ins Log (über den Logger)

### **ipump::IniFile::Error**

**private «enumeration» Class:** Mögliche Fehlerzustände

#### *ipump::IniFile::Error Attributes*

Attribute	Type	Notes
INI_FILE_NOT_FOUND «enum»	public :	
UNABLE_TO_READ_INI_FILE «enum»	public :	
INVALID_VALUE «enum»	public :	

### **ipump::InitControl**

**public Class:** Steuert die Initialisierungsphase. Der Parameter <code>initMode</code> in der INI-Datei bestimmt das Verhalten der Klasse. Es gibt zwei Modi:

- <ol>
    - <li>1 - Normal</li>
    - <li>2 - Mit Musik ("The Eye of the Tiger" - Rocky)</li>
- </ol> Der zweite Modus ist für das Projekt-Praktikum als Spielerei zu verstehen, um die Präsentation etwas lebendiger zu machen. Dieser Modus ist in vielerlei Hinsicht nicht ausgereift, so fehlt z.B. grösstenteils die Möglichkeit, die Initialisierung mit dem Reset-Knopf zu unterbrechen. Für die Bewertung soll deshalb nur erste Modus einbezogen werden.

#### *ipump::InitControl Attributes*

Attribute	Type	Notes
-----------	------	-------

logger	private : <i>Logger</i>	Referenz auf das Singleton Logger
parameter	private : <i>Parameter</i>	Referenz auf das Singleton Parameter
tankSystem	private : <i>TankSystem</i>	Referenz auf das Tank-System aus dem SITAS-API.
bottomTank	private : <i>sitas.client.Tank</i>	Referenz auf den unteren Tank aus dem SITAS-API. Der Zusatz 'sitas.client' ist nowendig, weil auch im Package 'ipump' eine Klasse 'Tank' existiert.
leftTank	private : <i>sitas.client.Tank</i>	Referenz auf den linken Tank aus dem SITAS-API. Der Zusatz 'sitas.client' ist nowendig, weil auch im Package 'ipump' eine Klasse 'Tank' existiert.
rightTank	private : <i>sitas.client.Tank</i>	Referenz auf den rechten Tank aus dem SITAS-API. Der Zusatz 'sitas.client' ist nowendig, weil auch im Package 'ipump' eine Klasse 'Tank' existiert.
BLINKTIME	private const : <i>int</i>	Nur für <code>initMode = 2</code> (mit Musik): Dauer des Aufblinkens einer einzelnen Lampe. Initial Value: 100;
FULLTIME	private const : <i>int</i>	Nur für <code>initMode = 2</code> (mit Musik): Dauer des Anzündens aller Lampen Initial Value: 2100;
SHORT_PAUSE	private const : <i>int</i>	Nur für <code>initMode = 2</code> (mit Musik): Dauer der kurzen Pause Initial Value: 290;
LONG_PAUSE	private const : <i>int</i>	Nur für <code>initMode = 2</code> (mit Musik): Dauer der langen Pause. Initial Value: 1200;
LEFT_RIGHT	private const : <i>int</i>	Nur für <code>initMode = 2</code> (mit Musik): Dauer der Pause zwischen einer links/rechts Blink-Sequenz Initial Value: 538;
soundFile	private : <i>String</i>	Nur für <code>initMode = 2</code> (mit Musik): Dateiname der Sound-Datei, welche während der Initialisierung abgespielt wird. Muss im gleichen Verzeichnis wie die Class-Datei liegen Initial Value: "startup.wav";
soundURL	private : <i>URL</i>	Nur für <code>initMode = 2</code> (mit Musik): URL auf die Sound-Datei, welche während der Initialisierung abgespielt wird. Wird benötigt, um die Sound-Datei zu laden.

#### *ipump*::InitControl Methods

Method	Type	Notes
InitControl ()	public:	Konstruktor. Erstellt Referenzen auf die Singletons Logger und Parameter, generiert die notwendigen Systemteile mit dem SITAS-API und lädt die Sound-Datei.
run ()	public: <i>boolean</i>	Startet die Initialisierung des Tank-Systems. Diese kann aus einem beliebigen Zustand her erfolgen. Nach einer erfolgreichen Initialisierung hat das System folgenden Zustand: <ul style="list-style-type: none"> <li>&lt;ul&gt;</li> <li>&lt;i&gt;Alles Wasser ist im TopTank&lt;/i&gt;</li> <li>&lt;i&gt;Alle anderen Tanks sind leer&lt;/i&gt;</li> <li>&lt;i&gt;Alle Lampen sind ausgeschaltet&lt;/i&gt;</li> <li>&lt;i&gt;Alle Ventile sind geschlossen&lt;/i&gt;</li> <li>&lt;i&gt;Die Pumpe ist ausgeschaltet&lt;/i&gt;</li> </ul> </ul> Der Modus 2 (Mit Musik) ist eine Spielerei und nicht für den produktiven Einsatz gedacht.
fillTopTank ()	private: <i>boolean</i>	Oberer Tank füllen. Bei <code>initMode = 2</code> (mit Musik) wird auch eine links/rechts Blinksequenz ausgeführt. Die Pumpe läuft so lange, bis der untere Tank leer ist. @return <code>false</code> wenn der Reset-Knopf gedrückt wurde, sonst <code>true</code>
playSequence ()	private: <i>void</i>	Spielt eine Lampen-Sequenz im Takt von "They eye of the tiger".
blink ( <i>Lamp</i> )	private: <i>void</i>	param: lamp [ Lamp - in ]

		Gibt an, welche Lampe anzuzünden ist.
		Leuchtet eine Lampe für eine kurze Zeit auf.
allLamps ( <i>int</i> )	private: <i>void</i>	param: duration [ int - in ] Dauer in Millisekunden  Zündet alle Lampen für die angegebene Dauer an.
playLeftRight ()	private: <i>void</i>	Blinkt abwechselungsweise mit den linken und den rechten Lampen. @param count Anzahl der Blinkzyklen
sleep ( <i>int</i> )	private: <i>void</i>	param: duration [ int - in ] Dauer in Millisekunden  Wartet für die angegebene Dauer.

## ipump::InitControl::Lamp

**public Class:** Mögliche Lampen

### ipump::InitControl::Lamp Attributes

Attribute	Type	Notes
LEFT_UPPER «enum»	public :	
LEFT_LOWER «enum»	public :	
RIGHT_UPPER «enum»	public :	
RIGHT_LOWER «enum»	public :	

## ipump::IPump

**public Class:** Startet das Programm iPump (enthält Main-Klasse).

### ipump::IPump Methods

Method	Type	Notes
main ( <i>String[]</i> )	public static: <i>void</i>	param: args [ String[] - in ] Übergebene Kommandozeilen-Parameter. Werden nicht verwendet.  Main-Klasse, startet das Programm.

## ipump::Lamp

**public Class:** Hat die Funktionalität für eine Lampe. Wird die Lampe eingeschaltet, löscht sie selbständig nach der im Parameter Objekt bestimmten Zeit wieder aus. Das Objekt muss periodisch über die Methode runStateMachine() aufgerufen werden.

### ipump::Lamp Attributes

Attribute	Type	Notes
myname	private : <i>SystemElement</i>	Name des Objektes, wird zu steuern der Lampe gebraucht und zum loggen
systemTimer	private : <i>SystemTimer</i>	Referenz auf ein Timer Objekt
leftTank	private : <i>sitas.client.Tank</i>	Referenz auf ein Stitas Tank. Wird zum Ansteuern der Lampen gebraucht.

rightTank	private : <i>sitas.client.Tank</i>	Referenz auf ein Stitas Tank. Wird zum Ansteuern der Lampen gebraucht.
logger	private : <i>Logger</i>	Referenz auf das Logger Objekt (Singelton) Initial Value: <i>Logger.getInstance()</i> ;
parameter	private : <i>Parameter</i>	Referenz auf das Parameter Objekt Initial Value: <i>Parameter.getInstance()</i> ;
tankSystem	private : <i>TankSystem</i>	Referenz auf das Sitas Client TankSystem Initial Value: <i>parameter.getTankSystem()</i> ;

### *ipump::Lamp Methods*

Method	Type	Notes
Lamp ( <i>SystemElement</i> )	public:	param: name [ <i>SystemElement</i> - in ] Damit das Objekt weiss welche Lampe es ist.  Konstruktor schaltet die Lampe aus
runStateMachine ()	public: <i>void</i>	wenn die Statemachine aufgerufen wird, schaut diese ob der Systemtimer abgelaufen ist und die Lampe ausgeschaltet werden muss Gibt keinen Parameter zurück, weil der Status der Lampe ausserhalb dieser Methode nicht interessiert
turnOff ()	public: <i>void</i>	Schaltet die Lampe aus.
turnOn ()	public: <i>void</i>	Schaltet die Lampe ein.

## OFF

*public State:*

## ON

*public State:*

### <anonymous>

*public Initial State:*

### **ipump::Logger**

*public Class*

**Implements: ILogger.** : Diese Klasse implementiert die Logger-Funktionalität. Die notwendigen Parameter holt sie sich von der Parameter-Klasse. Diese Klasse implementiert das Singleton-Pattern.

### *ipump::Logger Attributes*

Attribute	Type	Notes
logger	private static : <i>Logger</i>	Instanz des Logger, gebraucht für Singleton Pattern
databaseLogger	private : <i>DatabaseLogger</i>	Referenz auf DatabaseLogger
fileLogger	private : <i>FileLogger</i>	Referenz auf FileLogger
parameter	private : <i>Parameter</i>	Referenz auf Parameter

### *ipump::Logger Methods*

Method	Type	Notes
Logger ()	private:	Privater Konstruktor, wird nur einmal aufgerufen, wenn das erste mal geloggt werden soll. Diese Methode holt sich die notwendigen Parameter von der Parameterklasse.
getInstance ()	public static:	Gibt eine Referenz auf die Instanz dieses Objektes zurück. Falls

	<i>Logger</i>	die Instanz noch nicht existiert, wird eine instantiiert. @return Referenz auf Logger (einmalig)
writeLog ( <i>LogType</i> , <i>SystemElement</i> , <i>String</i> )	public: void	param: type [ LogType - in ] enthält den Nachrichtentyp param: source [ SystemElement - in ] gibt die Quelle der Nachricht an param: text [ String - in ] enthält die Nachricht  Übergibt die Log-Nachricht dem FileLogger und dem DatabaseLogger, sofern das Loggin im ini-File aktiviert wurde. Falls eine Meldung vom Typ 'ERROR' geloggt werden soll, wird sie auch unmittelbar auf der Konsole ausgegeben. @return void

## ipump::LogType

**public «enumeration» Class:** @value EVENT wird gebraucht für Ereignisse wie tank half full  
@value ACTION wird gebraucht für Aktionen wie leftTank.setFill()  
@value MESSAGE wird für alle anderen Log Einträge verwendet, die nicht in einer der anderen Kategorien passen  
@value ERROR wird für Fehlermeldungen verwendet

### ipump::LogType Attributes

Attribute	Type	Notes
EVENT «enum»	public :	
ACTION «enum»	public :	
MESSAGE «enum»	public :	
ERROR «enum»	public :	

## ipump::Parameter

**public Class:** Alle System-Parameter werden durch diese Klasse zur Verfügung gestellt. Die Klasse ist nach dem Singleton-Pattern implementiert, um sicherzustellen, dass immer höchstens eine Instanz der Klasse besteht. Um eine eigene Instanz zu erhalten, muss die statische Methode getInstance() verwendet werden.

### ipump::Parameter Attributes

Attribute	Type	Notes
userInterfaceActive	private : <i>boolean</i>	Wenn dieser Parameter auf <code>false</code> gesetzt ist, wird das User-Interface nicht angezeigt. Initial Value: true;
cycleCountLeftTank	private : <i>int</i>	Vorbgabewert für die Anzahl Zyklen des linken Tanks. Nur relevant, wenn userInterfaceActive <code>false</code> ist. Initial Value: 1;
cycleCountRightTank	private : <i>int</i>	Vorbgabewert für die Anzahl Zyklen des rechten Tanks. Nur relevant, wenn userInterfaceActive <code>false</code> ist. Initial Value: 1;
runCount	private : <i>int</i>	Vorbgabewert für die Anzahl der durchzuführenden Abläufe. Nur relevant, wenn userInterfaceActive <code>false</code> ist. Initial Value: 1;
lampTime	private : <i>int</i>	Vorbgabewert für die Anzünd-Dauer der Lampe (in Sekunden). Nur relevant, wenn userInterfaceActive <code>false</code> ist. Initial Value: 3;
pumpPause	private : <i>int</i>	Vorbgabewert für die Schondauer der Pumpe (in Sekunden). Nur relevant, wenn userInterfaceActive <code>false</code> ist. Initial Value: 8;

initMode	private : int	Modus für die Initialisierung. 1 = Normale Initialisierung, 2 = Initialisierung mit Musik Initial Value: 1;
sitasHost	private : String	Host des SITAS-Modells (<code>localhost</code>' für den Simulator) Initial Value: "localhost";
tankSystem	private : TankSystem	Referenz auf das Tank-System aus dem SITAS-API Initial Value: null;
database	private : String	Name der Datenbank für das Datenbank-Logging. Initial Value: "asademo";
databaseHost	private : String	Name des Datenbank-Host für das Datenbank-Logging. Initial Value: "localhost:2638";
databaseUserName	private : String	Benutzername der Datenbank für das Datenbank-Logging. Initial Value: "dba";
databaseUserPassword	private : String	Passwort des Benutzers der Datenbank für das Datenbank-Logging. Initial Value: "sql";
logActive	private : boolean	Wenn dieser Parameter auf <code>false</code> gesetzt ist, ist das Logging gänzlich deaktiviert. Initial Value: true;
logLevel	private : int	Gibt an, welche Art von Meldungen geloggt werden sollen: 1 = Nur Meldungen (Typ = <code>MESSAGE</code>) 2 = Alle Initial Value: 1;
logFilePath	private : String	Pfad für die Log-Datei. Initial Value: "C:\\Temp";
parameter	private static : Parameter	Referenz auf das Singleton Parameter. Diese Referenz wird über getInstance() zurückgegeben.

#### *ipump::Parameter Methods*

Method	Type	Notes
Parameter ()	private:	Der Konstruktor muss <code>private</code> sein (Singleton-Pattern).
getInstance ()	public static: Parameter	Gibt die Instanz der Klasse zurück. Es kann aufgrund des Singleton- Patterns nur eine Instanz geben, deshalb ist eine direkte Instanzierung mit <code>new</code> unzulässig. @return Parameter Instanz der Parameter-Klasse
setUserInterfaceActive (String)	«property set» public: void	param: in [ String - in ] <code>true</code> (als String!), wenn das User-Interface aktiv sein soll, irgendein anderer Wert, fall es deaktiviert werden soll.  Setter für <code>userInterfaceActive</code>.
isUserInterfaceActive ()	«property get» public: boolean	Getter für <code>userInterfaceActive</code>. @return boolean <code>true</code>, wenn das User-Interface aktiv ist.
setCycleCountLeftTank (String)	«property set» public: boolean	param: in [ String - in ] Anzahl Zyklen für den linken Tank.  Setter für <code>cycleCountLeftTank</code>.
getCycleCountLeftTank ()	«property get» public: int	Getter für <code>cycleCountLeftTank</code>. @return int Anzahl Zyklen für den linken Tank.
setCycleCountRightTank (String)	«property set» public: boolean	param: in [ String - in ] Anzahl Zyklen für den rechten Tank.  Setter für <code>cycleCountRightTank</code>.
getCycleCountRightTank ()	«property get» public: int	Getter für <code>cycleCountRightTank</code>. @return int Anzahl Zyklen für den rechten Tank.
setRunCount (String)	«property set» public: boolean	param: in [ String - in ] Anzahl der durchzuführenden Abläufe.  Setter für <code>runCount</code>.
getRunCount ()	«property get»	Getter für <code>runCount</code>.

	<code>public: int</code>	@return int Anzahl der durchzuführenden Abläufe.
<code>setLampTime (String)</code>	<code>«property set» public: boolean</code>	param: in [ String - in ] Brenndauer der Lampen in Sekunden.  Setter für <code>lampTime</code>.
<code>getLampTime ()</code>	<code>«property get» public: int</code>	Getter für <code>lampTime</code>. @return int Brenndauer der Lampen in Sekunden.
<code>setPumpPause (String)</code>	<code>«property set» public: boolean</code>	param: in [ String - in ] Schonpause der Pumpe in Sekunden.  Setter für <code>pumpPause</code>.
<code>getPumpPause ()</code>	<code>«property get» public: int</code>	Getter für <code>pumpPause</code>. @return int Schonpause der Pumpe in Sekunden.
<code>setInitMode (String)</code>	<code>«property set» public: boolean</code>	param: in [ String - in ] Init-Modus: 1 = Normale Initialisierung, 2 = Initialisierung mit Musik)  Setter für <code>initMode</code>.
<code>getInitMode ()</code>	<code>«property get» public: int</code>	Getter für <code>initMode</code>. @return int Init-Modus: 1 = Normale Initialisierung, 2 = Initialisierung mit Musik)
<code>setSitasHost (String)</code>	<code>«property set» public: void</code>	param: sitasHost [ String - in ] Host-Name vom SITAS-Modell, bzw. <code>localhost</code>, wenn der Simulator angesprochen werden soll.  Setter für <code>sitasHost</code>.
<code>getSitasHost ()</code>	<code>«property get» public: String</code>	Getter für <code>sitasHost</code>. @return String Aktueller Host vom SITAS-Modell, bzw. <code>localhost</code>, wenn der Simulator angeschlossen ist.
<code>setTankSystem (TankSystem)</code>	<code>«property set» public: void</code>	param: tankSystem [ TankSystem - in ] Instanz von <code>TankSystem</code>, von der SITAS-API.  Setter für <code>tankSystem</code>.
<code>getTankSystem ()</code>	<code>«property get» public: TankSystem</code>	Getter für <code>tankSystem</code>. @return TankSystem Instanz von <code>TankSystem</code>, von der SITAS-API.
<code>setDatabase (String)</code>	<code>«property set» public: void</code>	param: database [ String - in ] Name der Datenbank für das Logging.  Setter für <code>database</code>.
<code>getDatabase ()</code>	<code>«property get» public: String</code>	Getter für <code>database</code>. @return String Name der Datenbank für das Logging.
<code>setDatabaseHost (String)</code>	<code>«property set» public: void</code>	param: databaseHost [ String - in ] Hostname der Datenbank für das Logging.  Setter für <code>databaseHost</code>.
<code>getDatabaseHost ()</code>	<code>«property get» public: String</code>	Getter für <code>databaseHost</code>. @return String Hostname der Datenbank für das Logging.
<code>setDatabaseUserName (String)</code>	<code>«property set» public: void</code>	param: databaseUserName [ String - in ] Benutzername der Datenbank für das Logging.  Setter für <code>databaseUserName</code>.
<code>getDatabaseUserName ()</code>	<code>«property get» public: String</code>	Getter für <code>databaseUserName</code>. @return String Benutzername der Datenbank für das Logging.
<code>setDatabaseUserPassword (String)</code>	<code>«property set» public: void</code>	param: databaseUserPassword [ String - in ] Passwort des Datenbank-Benutzers für das Logging.  Setter für <code>databaseUserPassword</code>.
<code>getDatabaseUserPassword ()</code>	<code>«property get» public: String</code>	Getter für <code>databaseUserPassword</code>. @return String Passwort des Datenbank-Benutzers für das Logging.
<code>setLogActive (String)</code>	<code>«property set»</code>	param: in [ String - in ]

	<code>public: void</code>	Der Wert <code>"true"</code> (als String!) schaltet das Log ein, irgendein anderer String schaltet das Log aus.  Setter für <code>logActive</code> .
<code>isLogActive ()</code>	<code>«property get» public: boolean</code>	Getter für <code>logActive</code> . <code>@return boolean</code> Ist <code>true</code> , falls das Log eingeschaltet ist, ansonsten <code>false</code> .
<code>setLogLevel (String)</code>	<code>«property set» public: boolean</code>	param: in [ String - in ] Log-Level: 1 = Nur Meldungen (Typ = <code>MESSAGE</code> ), 2 = Alle (Typ = <code>MESSAGE</code> oder <code>WARNING</code> )  Setter für <code>logLevel</code> .
<code>getLogLevel ()</code>	<code>«property get» public: int</code>	Getter für <code>logLevel</code> . <code>@return int</code> Log-Level: 1 = Nur Meldungen (Typ = <code>MESSAGE</code> ), 2 = Alle (Typ = <code>MESSAGE</code> oder <code>WARNING</code> )
<code>setLogFilePath (String)</code>	<code>«property set» public: void</code>	param: <code>logFilePath</code> [ String - in ] Pfad für die Log-Datei.  Setter für <code>logFilePath</code> .
<code>getLogFilePath ()</code>	<code>«property get» public: String</code>	Getter für <code>logFilePath</code> . <code>@return</code> Pfad für die Log-Datei.
<code>writeToLog ()</code>	<code>public: void</code>	Schreibt alle aktuellen Parameter-Werte ins Log.
<code>toString ()</code>	<code>public: String</code>	(non-Javadoc) Gibt alle Werte des Objektes auf der Konsole aus. <code>@see java.lang.Object#toString()</code>

## ipump::SideTank

**public Class**

**Extends:** `Tank`. : Klasse der Seitentanks. Für den linken und rechten Seitentank wird je eine Instanz dieser Klasse gemacht. Diese Klasse bekommt von `TankControll` die Befehle füllen oder leeren. Und füllt oder leert selbstständig die Tanks. Dazu muss periodisch die `StateMachine` aufgerufen werden welche den Zustand des Tanks zurückgibt.

### ipump::SideTank Attributes

Attribute	Type	Notes
<code>topLamp</code>	<code>private : Lamp</code>	Referenz auf die obere Lampe
<code>bottomLamp</code>	<code>private : Lamp</code>	Referenz auf die untere Lampe
<code>state</code>	<code>private : TankState</code>	Attribut zum Speichern des Tankszustandes. Initialisierung auf <code>EMPTY</code> weil Tanks nach der Initialisierung leer sind. Falls sie nicht leer sind wird dies bei den ersten <code>StateMachine</code> aufrufen korrigiert. Initial Value: <code>TankState.EMPTY</code> ;
<code>filling</code>	<code>private : boolean</code>	Attribut zu speichern der Tätigkeit es Tanks. Wird für die Fehlerkorrektur der Sensoren gebraucht. <code>true</code> = Tank ist am füllen <code>false</code> = Tank ist am leeren Initial Value: <code>true</code> ;

### ipump::SideTank Methods

Method	Type	Notes
<code>SideTank (SystemElement)</code>	<code>public:</code>	param: <code>name</code> [ SystemElement - in ] Name des Tanks von Type <code>SystemElement</code>  Konstruktor Macht einen linken oder rechten Tank macht beide Ventile zu. braucht den Parameter damit er weiss ob er ein linker oder rechter Tank ist

runStateMachine ()	public: <i>TankState</i>	Statemachine der Seitentanks. Ein Seitentank hat nur 3 States: EMPTY, HALFFULL und FULL. Ein Seitentank hat keine eigene Logik. Die Tanks werden von der Klasse TankControl gesteuert. @return state Zustand des Tanks EMPTY, HALFFULL und FULL.
doEmpty ()	public: <i>void</i>	Leert den Tank. Stellt die Ventile. Zündet die untere Lampe an.
doFill ()	public: <i>void</i>	Füllt den Tank Stellt die Ventile. Zündet die obere Lampe an.
isEmpty ()	private: <i>boolean</i>	Gibt Anhand der Sensoren und des Tankzustandes (füllend oder leeren) den absoluten Zustand zurück. Logik der Fehlertoleranz siehe Dokumentation @return Tankzustand leer true wenn Tank leer ist
isFull ()	private: <i>boolean</i>	Gibt Anhand der Sensoren und des Tankzustandes (füllend oder leeren) den absoluten Zustand zurück. Logik der Fehlertoleranz siehe Dokumentation @return Tankzustand voll true wenn Tank voll ist
isHalfFull ()	private: <i>boolean</i>	Gibt Anhand der Sensoren und des Tankzustandes (füllend oder leeren) den absoluten Zustand zurück. Logik der Fehlertoleranz siehe Dokumentation @return Tankzustand halbvoll true wenn Tank halbvoll ist

## EMPTY

*public State:*

## FULL

*public State:*

## HALF\_FULL

*public State:*

## <anonymous>

*public Initial State:*

## ipump::SystemControl

*public Class:* Controller-Klasse für das Gesamtsystem. Initialisiert das System, ruft bei Bedarf das User-Interface auf und startet den Ablauf.

### ipump::SystemControl Methods

Method	Type	Notes
run ()	public: <i>void</i>	Startet den Controller.

## ipump::SystemElement

*public «enumeration» Class:* Enum-Klasse mit allen System-Bestandteilen (Klassen und physisches Modell).

### ipump::SystemElement Attributes

Attribute	Type	Notes
SYSTEM_CONTROL «enum»	public :	Repräsentiert die SystemControl Klasse.
INIT_CONTROL «enum»	public :	Repräsentiert die InitControl Klasse.
TANK_CONTROL	public :	Repräsentiert die TankControl Klasse.

<b>«enum»</b>		
USER_INTERFACE «enum»	public :	Repräsentiert die UserInterface Klasse.
INI_FILE «enum»	public :	Repräsentiert die IniFile Klasse.
PARAMETER «enum»	public :	Repräsentiert die Parameter Klasse.
LEFT_TANK «enum»	public :	Linker Tank des Modelles.
RIGHT_TANK «enum»	public :	Rechter Tank des Modelles.
BOTTOM_TANK «enum»	public :	Unterer Tank des Modelles.
LEFT_TANK_UPPER_VALVE «enum»	public :	Linkes oberes Ventil des Modelles.
LEFT_TANK_LOWER_VALVE «enum»	public :	Linkes unteres Ventil des Modelles.
RIGHT_TANK_UPPER_VALVE «enum»	public :	Rechtes oberes Ventil des Modelles.
RIGHT_TANK_LOWER_VALVE «enum»	public :	Rechtes unteres Ventil des Modelles.
BOTTOM_TANK_VALVE «enum»	public :	Ventil zwischen dem oberen und dem unteren Tank des Modelles.
LEFT_TANK_FULL_SENS OR «enum»	public :	Sensor am linken Tank. Zeigt an, wenn der Tank voll ist.
LEFT_TANK_HALF_SENS OR «enum»	public :	Sensor am linken Tank. Zeigt an, wenn der Tank halb-voll ist.
LEFT_TANK_EMPTY_SENSOR «enum»	public :	Sensor am linken Tank. Zeigt an, wenn der Tank leer ist.
RIGHT_TANK_FULL_SENSOR «enum»	public :	Sensor am rechten Tank. Zeigt an, wenn der Tank voll ist.
RIGHT_TANK_HALF_SENSOR «enum»	public :	Sensor am rechten Tank. Zeigt an, wenn der Tank halb-voll ist.
RIGHT_TANK_EMPTY_SENSOR «enum»	public :	Sensor am rechten Tank. Zeigt an, wenn der Tank leer ist.
LEFT_TANK_UPPER_LAMP «enum»	public :	Obere Lampe des linken Tanks.
LEFT_TANK_LOWER_LAMP «enum»	public :	Untere Lampe des linken Tanks.
RIGHT_TANK_UPPER_LAMP «enum»	public :	Obere Lampe des rechten Tanks.
RIGHT_TANK_LOWER_LAMP «enum»	public :	Untere Lampe des rechten Tanks.
PUMP «enum»	public :	Pumpe am Modell.

## ipump::SystemTimer

**public Class:** Ist ein "one shot timer". Wird gesetzt mit der Methode `setTimerAfterSeconds()`. Die Methode `isTimerExpired()` gibt True wenn der Timer abgelaufen ist.

### ipump::SystemTimer Attributes

Attribute	Type	Notes
endTime	private : <i>long</i>	Speichert die Systemzeit, zu der der Timer abläuft. Absolute Zeit in Millisekunden wann Timer abläuft. Endtime wird aus der aktuellen Zeit plus der setTime berechnet. Initial Value: 0;
timeractive	package : <i>boolean</i>	Zustand des Timers true wenn Timer am laufen ist. Initial Value: false;

### ipump::SystemTimer Methods

Method	Type	Notes
<code>isTimerExpired ()</code>	public: <i>boolean</i>	Ermittelt, ob die Zeit abgelaufen ist. Gibt True zurück wenn Timer abgelaufen ist. @return expired true wenn Timer abgelaufen ist. Kann nur einmal abgefragt werden. Danach kommt false bis zu nächsten mal Timersetzen.
<code>setTimerAfterSeconds (int)</code>	public: <i>void</i>	param: time [ int - in ] Zeit für Timer in Sekunden  Startet den Timer mit der Anzahl angegebenen Sekunden

## ipump::Tank

**public abstract Class:** Abstrakte Klasse eines Tanks

### ipump::Tank Attributes

Attribute	Type	Notes
myname	protected : <i>SystemElement</i>	Attribut welches den Namen das Objektes trägt. Wird gebraucht für das Logging und für die Ansteuerung der Ventile und Lampen.
logger	protected : <i>Logger</i>	Referenz auf das Logger Objekt (Singelton)
parameter	protected : <i>Parameter</i>	Referenz auf das Parameter Objekt (Singelton)
tankSystem	protected : <i>TankSystem</i>	Referenz auf das Sitas Client TankSystem (Singelton)
thistank	protected : <i>sitas.client.Tank</i>	Referenz auf den Sitas Client Tank.

### ipump::Tank Methods

Method	Type	Notes
<code>Tank (SystemElement)</code>	public:	param: name [ SystemElement - in ] Name des Tanks  Konstruktor Macht die grundätzlichen Sachen welche in den beiden abgeleiteten Klassen benötigt werden: Holt Referenzen auf die Singelton Instanzen von Logger, Parameter und tankSystem. Und setzt das Attribut myname.

## **ipump::TankControl**

**public Class:** Steuert einen kompletten Ablauf mit allen Zyklen. Die dazu notwendigen Daten werden aus dem Parameter-Singleton bezogen.

### *ipump::TankControl Attributes*

Attribute	Type	Notes
logger	private : <i>Logger</i>	Referenz auf das Singleton Logger
parameter	private : <i>Parameter</i>	Referenz auf das Singleton Parameter

### *ipump::TankControl Methods*

Method	Type	Notes
TankControl ()	public:	Konstruktor. Erstellt Referenzen auf die Singletons Logger und Parameter, erstellt die Referenzen auf die System-Komponenten und merkt sich die aktuellen Zyklus- und Ablauf-Werte aus dem Parameter-Singleton.
run ()	public: boolean	Gibt <code>true</code> zurück, wenn alles erfolgreich lief oder <code>false</code>, falls der Benutzer am Modell "Reset" gedrückt hat.

## **ipump::TankControl::TankControlState**

**private «enumeration» Class:** Eigener Status

### *ipump::TankControl::TankControlState Attributes*

Attribute	Type	Notes
FILLING_LEFT «enum»	public :	Der linke Tank wird gerade gefüllt
EMPTYING_LEFT «enum»	public :	Der linke Tank wird gerade geleert
FILLING_RIGHT «enum»	public :	Der rechte Tank wird gerade gefüllt
EMPTYING_RIGHT «enum»	public :	Der rechte Tank wird gerade geleert
EMPTYING_BOTH «enum»	public :	Beide Tanks werden gerade geleert

## **EMPTYING\_BOTH**

**public State:**

### *EMPTYING\_BOTH Methods*

Method	Type	Notes
leftTank.isEmpty() ()	public: entry	
rightTank.isEmpty() ()	public: entry	

## **EMPTYING\_LEFT**

**public State:**

### *EMPTYING\_LEFT Methods*

Method	Type	Notes
leftTank.isEmpty() ()	public: entry	

## **EMPTYING\_RIGHT**

*public State:*

### ***EMPTYING\_RIGHT Methods***

Method	Type	Notes
rightTank.isEmpty() ()	public: <i>entry</i>	

## **FILLING\_LEFT**

*public State:*

### ***FILLING\_LEFT Methods***

Method	Type	Notes
leftTank.isFilled() ()	public: <i>entry</i>	

## **FILLING\_RIGHT**

*public State:*

### ***FILLING\_RIGHT Methods***

Method	Type	Notes
rightTank.isFilled() ()	public: <i>entry</i>	

## **<anonymous>**

*public Final State:*

## **<anonymous>**

*public Initial State:*

## **ipump::TankState**

*public «enumeration» Class:*

### ***ipump::TankState Attributes***

Attribute	Type	Notes
FULL «enum»	public :	Tank ist voll
HALFFULL «enum»	public :	Tank ist halbvoll
EMPTY «enum»	public :	Tank ist leer

## **ipump::UserInterface**

*public Class:* Stellt das textbasierte UserInterface des Systems auf der Konsole dar.

### ***ipump::UserInterface Attributes***

Attribute	Type	Notes
parameter	private : <i>Parameter</i>	Referenz auf das Singleton Parameter

***ipump::UserInterface Methods***

<b>Method</b>	<b>Type</b>	<b>Notes</b>
UserInterface ()	public:	Konstruktor. Erstellt eine Referenz auf das Parameter-Singleton.
readParameter ()	public: <i>void</i>	Stellt das User-Interface auf der Konsole dar. Der Benutzer kann alle Parameter einzeln eingeben und/oder das Programm beenden. Die eingegebenen Parameter werden validiert. Falls ein ungültiger Wert eingegeben wurde, wird eine Fehlermeldung ausgegeben und die Eingabe kann wiederholt werden.
printLine ( <i>String</i> )	private: <i>void</i>	param: text [ String - in ] Auszugebender Text  Gibt einen Text auf der Konsole auf einer neuen Zeile aus.
printError ()	private: <i>void</i>	Gibt eine Fehlermeldung aus, dass ein Eingabewert unzulässig ist.

## 14. Testplan

Szenario / Testfall	Beschreibung Testfall	Erwartetes Testresultat	Resultat
SZ1 Normale Initialisierung	Initialisierung unter normalen Bedingungen. <ul style="list-style-type: none"><li>- Alles Wasser ist im unteren Tank</li><li>- Wasserstand ist normal</li><li>- Keine Flussbeschränkungen</li><li>- Die Pumpe funktioniert</li><li>- Alle Ventile funktionieren</li><li>- Alle Lampen funktionieren</li><li>- Alle Sensoren funktionieren</li></ul>	Die Initialisierung wird erfolgreich beendet. <ul style="list-style-type: none"><li>- Alles Wasser ist im oberen Tank</li><li>- Die Pumpe ist ausgeschaltet</li><li>- Alle Ventile sind geschlossen</li><li>- Alle Lampen sind ausgeschaltet</li></ul>	o.k.
SZ2 Start mit/ohne Logging	In derINI-Datei wird das Logging über den Parameter 'logActive' aktiviert bzw. deaktiviert.	Bei deaktiviertem Logging wird kein Log geschrieben (weder in der DB noch als Datei).	o.k.
SZ3 Initialisierung mit Flussbeschränkungen	Initialisierung mit Flussbeschränkungen bei der Verbindung der Seitentanks zum unteren Tank. <ul style="list-style-type: none"><li>- Linker und Rechter Tank ist halbvoll</li><li>- Wasserstand ist normal</li><li>- Die Leitung vom linken Seitentank zum unteren Tank ist halb geschlossen</li><li>- Die Leitung vom rechten Seitentank zum unteren Tank ist halb geschlossen</li><li>- Die Pumpe funktioniert</li><li>- Alle Ventile, Lampen und Sensoren funktionieren</li></ul>	Die Initialisierung wird erfolgreich beendet. <ul style="list-style-type: none"><li>- Alles Wasser ist im oberen Tank</li><li>- Die Pumpe ist ausgeschaltet</li><li>- Alle Ventile sind geschlossen</li><li>- Alle Lampen sind ausgeschaltet</li></ul>	o.k.
SZ4 Unterbruch durch Reset-Taste während Ablauf	Das System befindet sich an irgendeinem Punkt während eines gestarteten Ablaufes. Am SITAS-Modell wird die Reset-Taste betätigt.	- Der Ablauf wird beendet - Auf der Konsole wird eine Fehlermeldung angezeigt	o.k.
SZ5 Unterbruch durch Reset-Taste während Parametereingabe	Auf der Konsole wird die Parametereingabe angezeigt, der Ablauf ist noch nicht gestartet. Am SITAS-Modell wird die Reset-Taste betätigt.	- Die Parametereingabe wird weiterhin angezeigt - Es wird keine Fehlermeldung angezeigt - Nach vollständiger Eingabe wird der Ablauf gestartet und sofort beendet - Auf der Konsole wird eine Fehlermeldung angezeigt	o.k.
SZ6 Programm startet ohne Datenbankverbindung	Das Programm wird gestartet. Aus irgendwelchen Gründen (z.B. keine Datenbank vorhanden, Datenbank falsch eingerichtet, falsche Benutzerparameter) kann keine Verbindung zum Datenbankserver aufgebaut werden.	- Das Programm läuft fehlerfrei weiter. - Auf der Konsole wird ein Hinweis ausgegeben, dass kein Datenbank-Logging erfolgen kann	o.k.
SZ7 Initialisierung ohne Verbindung zum SITAS-Modell	Initialisierung ohne Verbindung zum SITAS-Modell	- Die Initialisierung wird nicht gestartet - Es wird eine Fehlermeldung "Verbindung zum SITAS-Modell konnte nicht aufgebaut werden" ausgegeben - Das Programm wird beendet	o.k.
SZ8 Normale Initialisierung, Wasser bereits im oberen Tank	Initialisierung unter normalen Bedingungen. <ul style="list-style-type: none"><li>- Alles Wasser ist im oberen Tank</li><li>- Wasserstand ist normal</li><li>- Keine Flussbeschränkungen</li><li>- Die Pumpe funktioniert</li><li>- Alle Ventile funktionieren</li><li>- Alle Lampen funktionieren</li><li>- Alle Sensoren funktionieren</li></ul>	Die Initialisierung wird erfolgreich beendet. <ul style="list-style-type: none"><li>- Alles Wasser ist im oberen Tank</li><li>- Die Pumpe ist ausgeschaltet</li><li>- Alle Ventile sind geschlossen</li><li>- Alle Lampen sind ausgeschaltet</li></ul>	o.k.

Szenario / Testfall	Beschreibung Testfall	Erwartetes Testresultat	Resultat
SZ9 Normaler Ablauf	Normaler Ablauf  Variable Wert Beschreibung cycleCountLeftTank 3 Anzahl linker Zylen cycleCountRightTank2 Anzahl rechter Zylen runCount 2 Anzahl Abläufe lampTime 8 Zeit, wie lange eine Lampe leuchtet pumpPause 10 Wartezeit der Pumpe	Normaler Ablauf wie Timingdiagramm spezifiziert. Ablauf entspricht dem Szenario 1 (Beschrieben im Kapitel Szenarien)  Die Lampen löschen nach 8 Sekunden ab. Das ist bei offenen Ventilen nach ca. halber Füllung oder Leerung der Fall.	o.k.
SZ10 Ablauf mit Zu- und Abflussbeschränkungen	Der linke Tank ist aufgrund von Zu- und Abflussbeschränkungen 5x langsamer als der rechte Tank.  Variable Wert Beschreibung cycleCountLeftTank 3 Anzahl linker Zylen cycleCountRightTank2 Anzahl rechter Zylen runCount 2 Anzahl Abläufe lampTime 8 Zeit, wie lange eine Lampe leuchtet pumpPause 10 Wartezeit der Pumpe	Normaler Ablauf wie Timingdiagramm spezifiziert. Keine Timouts oder Laufzeitprobleme. Das Modell funktioniert normal.  Ablauf entspricht dem Szenario 2 (Beschrieben im Kapitel Szenarien)  Die Lampen löschen nach 8 Sekunden ab. Das ist bei offenen Ventilen nach ca. halber Füllung oder Leerung der Fall.	o.k.
SZ11 Normaler Ablauf mit sehr kurzer lampTime	Normaler Ablauf mit sehr kurzer lampTime  Variable Wert Beschreibung cycleCountLeftTank 3 Anzahl linker Zylen cycleCountRightTank2 Anzahl rechter Zylen runCount 2 Anzahl Abläufe lampTime 1 Zeit, wie lange eine Lampe leuchtet pumpPause 10 Wartezeit der Pumpe	Normaler Ablauf wie Timingdiagramm spezifiziert. Keine Timouts oder Laufzeitprobleme. Das Modell funktioniert normal.  Ablauf entspricht dem Szenario 3 (Beschrieben im Kapitel Szenarien)  Die Lampen löschen nach 1 Sekunden ab.	o.k.
SZ12 Ablauf mit sehr langer pumpPause	Es wird ein Ablauf mit einer sehr langen pumpPause (100 Sekunden) gestartet.	Der Ablauf wird korrekt durchgeführt. Es entstehen lange Wartezeiten, sobald der obere Tank leer ist, da zuerst die Schonzeit der Pumpe abgewartet werden muss.	o.k.
SZ13 Ablauf mit defekten Sensoren	Die Sensoren sind defekt: - haben ein Wasserblase -> sie zeigen immer Wasser an - haben ein Luftblase -> zeigen immer kein Wasser an  Die Fehlerkorrektur muss so funktionieren wie in der Doku spezifiziert.	Weil dieser Test auf dem Modell nicht durchführbar. Darum wurde dieses Funktion Softwaremäßig mit speziellen Testfunktionen getestet.  (Siehe Kapitel Sensor Fehlerkorrektur)	o.k.
SZ14 Fehleingaben am User-Interface	Bei der Eingabe der Parameterwerte am User-Interface werden nicht zulässige Werte eingegeben (z.B. 0 für die Schonzeit der Pumpe).	Pro fehlerhaftem Parameter wird eine Fehlermeldung eingegeben. Die Eingabe wird so lange wiederholt, bis ein korrekter Wert eingegeben wurde.	o.k.
SZ15 Integer-Überlauf am User-Interface	Am User-Interface wird für einen Parameter, der nur Integer-Werte entgegennehmen kann (z.B. Schonzeit der Pumpe) ein Wert eingegeben, der den zulässigen Wertebereich für Integer ( $2^{147'483'647}$ ) übersteigt.	Pro fehlerhaftem Parameter wird eine Fehlermeldung eingegeben. Die Eingabe wird so lange wiederholt, bis ein korrekter Wert eingegeben wurde.	o.k.

## 15. Timingdiagramm

Um das genaue Verständnis für die Aufgabenstellung zu erhalten, haben wir vorgängig ein Timingdiagramm erstellt, das den Ablauf eines Beispiefalles zeigt.

Dieses Diagramm wurde vor dem Beginn des Designs mit dem Auftraggeber besprochen um das Verständnis zu prüfen.

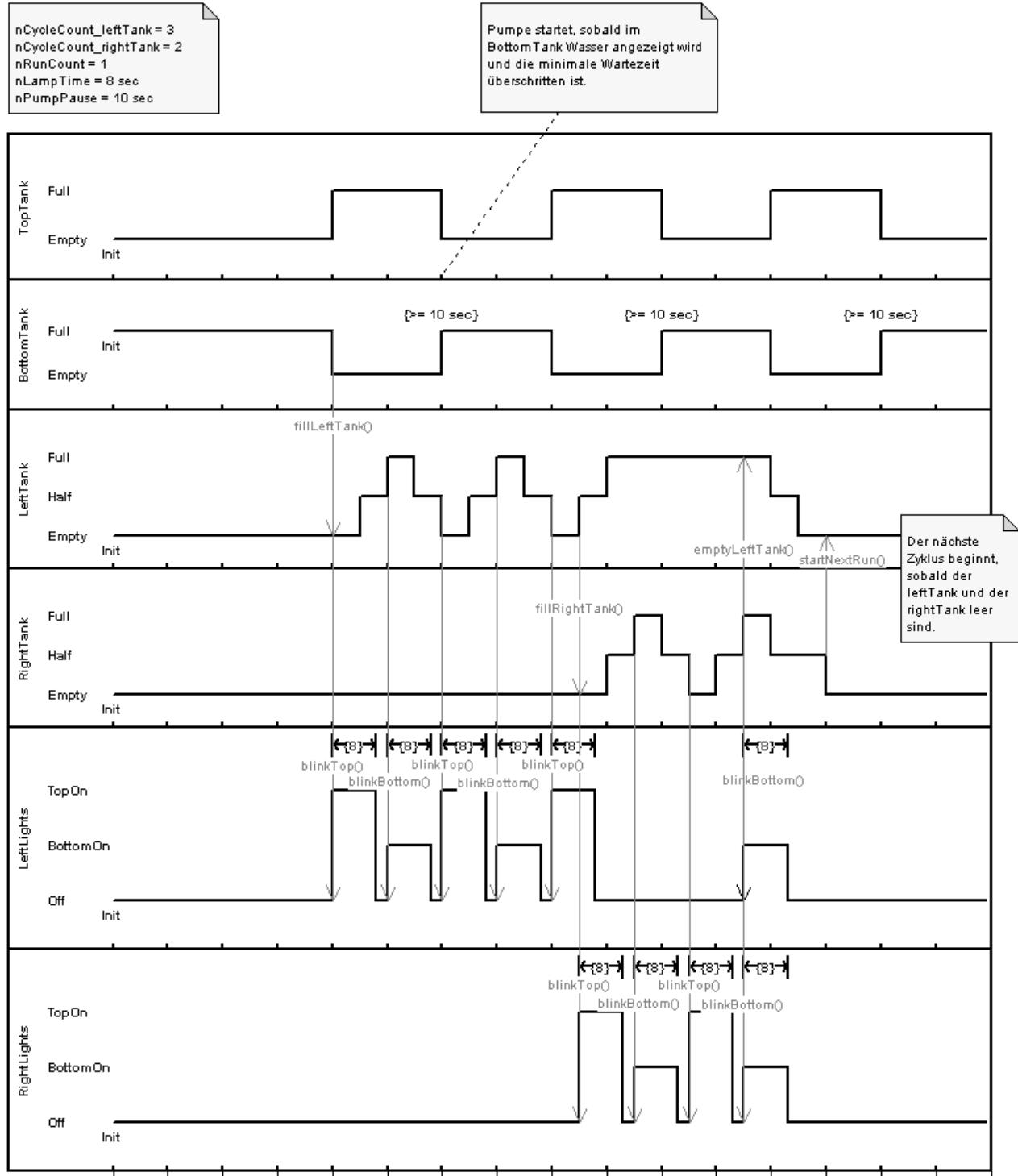


Illustration 18: Timing-Diagramm

# **16. Bedienungsanleitung**

Bitte diesen Abschnitt ganz durchlesen, damit keine Schäden am SITAS Modell entstehen.

## **16.1. Vorbereitungen**

### **16.1.1 Datenbank (optional)**

iPump unterstützt neben dem FileLogging auch noch ein DatenbankLogging. Dazu muss eine Datenbank installiert werden und eine gültige Tabellendefinition vorhanden sein.

iPump wurde mit Sybase Database entwickelt, es können deshalb keine Garantien gewährleistet werden, dass es auch mit andern Datenbanksystem funktioniert.

Eine Vollversion von der ASA-Datenbank kann unter

<http://www.sws.bfh.ch/~schmd/db/DOWNLOADS/PRODUKTE/ASA/ASA%209.zip>

heruntergeladen werden.

Die Installation sollte keine weiteren Einstellungen benötigen, nach erfolgreicher Installation muss noch die nötige Tabellendefinition für iPump eingegeben werden:

```
create table iPumpLog
(
    idPumpLog      numeric(8, 0) not null default autoincrement,
    logTimeStamp   timestamp not null default timestamp,
    logType        varchar(10),
    logSource       varchar(20),
    logMessage      varchar(100),
    primary key     (idPumpLog)
);
```

Wird diese Datenbank verwendet benötigt das keine weiteren Einstellungen oder Anpassungen. Nun können Sie die iPump mit DatenbankLogging verwenden.

### **16.1.2 Konfiguration ini-File (iPump.ini)**

Im beiliegenden IniFile können Sie die folgende Parameter konfigurieren:

- userInterfaceActive = [true | false]  
mit diesem Parameter können Sie das User Interace deaktivieren und iPump verwendet die angegebenen Werte, die Sie weiter unten im File finden.
- initMode = [1 | 2]  
1 = Einfache Initialisierung, 2 = Initialisierung mit Musik
- SitasHost = [sitas0 | sitas1 | sitas2 | localhost]  
mit sitasN bestimmen Sie das gewünschte SITAS Model, localhost für die Simulation
- database = [name]  
Datenbankname
- databaseHost =[host address]  
Hostadresse der Datebank
- databaseUserName = [user name]  
Benutzername für die Datenbank
- databaseUserPassword = [password]  
Benutzerpasswort für die Datenbank
- logActive = [true | false]  
Aktivieren/Deaktivieren des Logging
- logLevel = [1 | 2]  
Filter für die LogMeldungen 1 = alle, 2 = nur Fehler
- logFilePath = [file path]  
Pfadangabe für dasLogFile

Nachfolgende Parameter sind nur aktiv, wenn das User Interface deaktiviert ist, (userInterfaceActive = false)

- cycleCountLeftTank = [1, n]  
Anzahl Füll/Leer-Zyklen des linken Tanks
- cycleCountRightTank = [1, n]  
Anzahl Füll/Leer-Zyklen des rechten Tanks
- runCount = [1, n]  
Anzahl Sequenzdurchläufe
- lampTime = [1, n]  
Einschaltzeit [s] der Signallampen an den beiden Seitentanks
- pumpPause = [1, n]  
Schutzzzeit [s] für die Pumpe im BottomTank

Damit Sie iPump mit Ihrem modifizierten ini-File laufen lassen können, müssen Sie das gesamte Programmverzeichnis iPump auf Ihren lokalen Datenträger kopieren, damit Sie anschliessend Ihr verändertes ini-File abspeichern können.

## 16.2. iPump starten

Stellen Sie eine Verbindung mit dem gewünschten SITAS-Modell her, dazu müssen Sie unter Umständen das ini-File anpassen. Haben Sie Probleme mit der Verbindungsherstellung, kontaktieren Sie die Bedienungsanleitung über das SITAS.

iPump benötigt keine Installation, sondern nur eine JRE (Java Runtime Environment) Installation ab Version 5.0. Um iPump zu starten, passen Sie den Pfad in der Datei start\_iPump.bat an und starten Sie danach die Datei start\_iPump.bat in der Konsole.

## 16.3. Bedienung

Bei der Entwicklung von iPump wurde darauf geachtet, dass die Bedienung möglichst intuitiv und äusserst einfach ist. Deshalb ist es sehr einfach, iPump zu bedienen.

Nach dem Ausführen des iPump Start-Befehls erscheint folgende Anzeige und iPump beginnt mit der Initialisierungssequenz:

```
...: iPump Version 1.0 :...
Warning: Keine Datenbank gefunden!
Datenbank logging deaktiviert!
Initialisierung wird gestartet.
Initialisierung ist beendet.
```

Ist die Initialisierung abgeschlossen werden Sie aufgefordert die Ablaufparameter einzugeben. Es ist nicht möglich falsche Eingaben zu machen. Nachdem alle geforderten Parameter eingegeben sind, startet iPump automatisch die von Ihnen programmierte Sequenz.

```
Anzahl Zyklen linker Tank (1-n):
hallo
Dieser Eingabewert ist unzulässig.
Anzahl Zyklen linker Tank (1-n):
1
Anzahl Zyklen rechter Tank (1-n):
2
Anzahl Abläufe (1-n):
3
Brenndauer der Lampen in Sekunden (1-n):
4
Schondauer der Pumpe in Sekunden (1-n):
5
Ablaufkontrolle wird gestartet.
```

Möchten Sie den Ablauf frühzeitig abbrechen drücken Sie die Reset-Taste am SITAS-Modell.  
Nach dem Ablauf werden Sie gefragt, ob Sie iPump beenden möchten.

```
Ablaufkontrolle ist beendet.
Möchten Sie iPump beenden (J/N):
```

Wenn Sie iPump noch einmal mit anderen Parameter laufen lassen möchten drücken Sie die 'N'-Taste, um iPump zu Beenden drücken Sie die 'J'-Taste.

# **17. Gemachte Erfahrungen**

## **17.1. Dokumentation mit OpenOffice**

OpenOffice unterstützt Master- und Subdokumente, was das Arbeiten an verschiedenen Teilen der Dokumentation ermöglicht. Jedoch ist das Handling mit den Formatvorlagen etwas kompliziert, aber das kennen wir von MS Word ja auch! Als Alternative zum Bezahl-Word ist es jedoch allemal geeignet.

## **17.2. Codeverwaltung mit TortoiseSVN und GoogleCode**

Als Hilfsmittel zur Codeverwaltung haben wir TortoiseSVN mit GoogleCode eingesetzt. Tortoise SVN ist ein Client für das Codeverwaltungssystem CVS. GoogeCode ist ein kostenlos zugänglicher CVS-Server.

Diese Kombination ist eine einfache und kostenlose Möglichkeit, Text- und binäre files von verschiedenen Rechnern/Clients zentral zu verwalten. Der Aufwand für die Installation der benötigten Tools hat sich unserer Meinung nach ausbezahlt durch die erhöhte Sicherheit und einfachere Zusammenarbeit, insbesondere während der Phase der verteilten Entwicklung.

## **17.3. CASE-Tool Enterprise Architect**

Wir haben Enterprise Architect (EA) als OOSE-Tool eingesetzt. Dies ist ein sehr mächtiges Tool, und der grosse Teil der Designdoku stammt direkt von der Dokumentationsfunktion von EA.

Aus Gründen der Einfachheit und möglicher Probleme beim Mergen haben wir auf die Master/Replica Möglichkeit für die Teamarbeit verzichtet. Dies bedingte jedoch, dass strikte nur eine Person am Modell arbeitete was sich schlussendlich aber doch als etwas problematisch herausstellte.

EA bietet auch verschiedene Formen vor Reverse-Engineering an.

## **17.4. Design-Entscheide festhalten**

Während der Design-Phase müssen laufend Entscheidungen getroffen werden. Es wäre gut gewesen, die wichtigsten Design-Entscheide schriftlich festzuhalten, damit zu einem späteren Zeitpunkt darauf zurückgegriffen werden kann. Dies vermeidet unnötige Diskussionen.

## **17.5. Codegenerierung mit EA**

### **Exakte Umsetzung des Designs**

Die Generierung des Codes gestaltete sich recht einfach und hatte den grossen Vorteil, dass die vereinbarten Klassen- und Methodennamen inkl. Signaturen auch wirklich dem Design entsprachen. Allerdings wurde schnell klar, dass man in der Designphase nicht an jedes kleinste Detail denken kann. Dadurch mussten auch am generierten Code noch ein paar Anpassungen gemacht werden.

Bei der Generierung des Code-Gerüsts aus EA gab aber auch diverse Schwierigkeiten:

### **Die Namen der Members (Assoziationen) war nicht kleingeschrieben**

Um dies zu erreichen, haben wir für jede Assoziation bei der Target-Role den Rollennamen mit Kleinbuchstaben beginnend aufgeschrieben.

### **Die Festlegung eines Konstruktors ist offenbar nicht möglich**

Wenn man eine statische Klassenmethode mit dem Namen der Klasse erstellt, wird bei der Codegenerierung eine eigene Methode gemacht, die „void“ als Rückgabewert hat. Dies erlaubt der Compiler natürlich nicht.

### **Code-Modell-Synchronisation**

Nach Abschluss der Entwicklungsphase haben wir mit der Reverse-Engineering Funktion die Änderungen am Code wieder mit dem Klassenmodell synchronisiert. Dies hat problemlos funktioniert. Leider wurde durch den Re-Import das Layout des Klassenmodells verunglimpft und musste nochmals bereinigt werden.

## **17.6. Codierung**

### **Verantwortlicher pro Klasse**

Wir haben uns entschieden, nach der Code-Generierung aus Enterprise Architect parallel zu entwickeln. Dazu haben wir jede Klasse einem Verantwortlichen zugewiesen. Dieses Verfahren hat sich eigentlich gut bewährt und wir kamen rasch voran. Man muss sich natürlich davor hüten, einen kleinen Fehler in einer "fremden" Klasse mal schnell selbst zu beheben... Das würde schnell zu chaotischen Zuständen führen.

### **Klassennamen mit SITAS koordinieren**

Wir haben einer Klasse den gleichen Namen 'Tank' gegeben wie eine Klasse im sitas.client package. Dies hat bei der Codierung zu etwas Verwirrung geführt.

### **Fachklassen mit SITAS koordinieren**

Wir haben für die Lampe eine eigene Klasse gemacht. Dies ist unterschiedlich zur sitas.client Architektur wo die Lampen dem linken oder rechten Tank zugeordnet sind. Dies führte bei der Codierung dazu dass in der Klasse Lampe auch noch je ein linker und rechter Tank von stas.client.Tank geholt werden musste mit der Methode TankSystem.getTank()).

### **Zeitaufwand fürs Codieren klein**

Das Codieren haben wir zeitlich etwas überschätzt. Es ging einfacher und schneller als erwartet und man hätte noch etwas länger am Design arbeiten können.

## **17.7. Testen**

Wir sind erstaunt, wie wenig Fehler wir am Schluss noch zu beheben hatten. Durch die lange Designphase hatten wir doch recht wenig Zeit ins Codieren und Bugfixen stecken müssen.

## 18. Glossar

Ablauf	Ein Ablauf besteht aus 1-n Zyklen für den linken und 1-n Zyklen für den rechten Seitentank.
EAP	Dateityp von Enterprise Architect (ein CASE-Tool) der ein ganzes Projekt beinhaltet. Manchmal auch als Kurzbezeichnung für Enterprise Architect selbst verwendet.
LogType	Der LogType spezifiziert die dazugehörige Log-Nachricht, es können folgende Nachrichtentypen auftreten:
	<ul style="list-style-type: none"><li>• ACTION deklariert Aktionen welche von iPump ausgelöst werden (z.B. leftTank.doFill())</li><li>• EVENT ist eine Meldung welche vom SITAS (z.B. leftTank = FULL)</li><li>• ERROR ist eine Fehlermeldung</li><li>• MESSAGE wird für alle anderen Nachrichten verwendet, welche nicht auf einen vorhergehenden Typ passen</li></ul>
nCycleCount_leftTank	Anzahl Füllungs-/Leerungs-Zyklen für den linken Tank [1..n]
nCycleCount_rightTank	Anzahl Füllungs-/Leerungs-Zyklen für den rechten Tank [1..n]
nLampTime	Leuchtdauer der Lampen in Sekunden [1..n]
nPumpPause	Ausschaltzeit der Pumpe, nachdem der untere Tank leer geworden ist, in Sekunden [1..n]
nRunCount	Anzahl Abläufe (Zyklen des linken und rechten Tanks) [1..n]
SITAS	SITAS ist eine Abkürzung für "Simple Tank System" (einfaches Behälter-System). Es besteht im wesentlichen aus vier Tanks, einer Pumpe, vier Lampen, Ventilen und einer Steuerungselektronik, die eine Steuerung des Systems über ein Java-API ermöglicht.
	Die Tanks umfassen:
	<ul style="list-style-type: none"><li>• Den unteren Tank (100% Füllvermögen)</li><li>• Den oberen Tank (100% Füllvermögen)</li><li>• Einen linken Seitentank (50% Füllvermögen)</li><li>• Einen rechten Seitentank (50% Füllvermögen)</li></ul>
	Der untere und der obere Tank sind über eine Pumpe miteinander verbunden.
Zyklus	Ein Zyklus besteht aus einem vollständigen Füllen und Leeren eines Seitentanks.