

VS - Projekt SmartHome - Server

Projektteam

- Raphael AMANN sa19b001
- Maria SCHILDBECK sa19b035
- Emil SEDLACEK sa19b501
- Daniela WEINHAPPL sa19b011

SS2021 / BSA4 / FHTW / 22.06.2021

Inhaltsverzeichnis

Prephase	3
Skills	3
Arbeitsteilung, verwendete Tools & Technologien	3
Meilensteine	3
Beschreibung der APIs	5
UI - W3C	5
Überlegungen & Fortschritt	5
Funktion und Programmierung	5
Hauptseite	5
Unterseite Licht	6
Unterseite Jalousien	7
Lessons Learned	8
SmartHome Server - REST	9
Überlegungen	9
Funktion und Programmierung	9
REST API functions	10
Lessons Learned	11
Wetter Service - SOAP	12
Überlegungen	12
Funktion und Programmierung	12
Anfrage-Antwort-Aufbau	13
Lessons Learned	15

Überlegungen

Skills

A - gut, B - mittelmäßig, C - weniger gut

Name	Java	.NET (C#)	Python	W3C (PHP, HTML, CSS, JS)
Daniela	A	B	C	A
Emil	A	B	A	B
Maria	B	B	A	B
Raphael	A	C	B	A

Arbeitsteilung, Verwendete Tools & Technologien

Service	Technologie	Sprache / IDE	Deployment	Zuteilung (Verantw., Mitarb.)
Wetter Service	SOAP	Java / Eclipse JEE	OnPremise - Emil	Emil, Daniela & Maria
Smart Home Service	REST	Python FLASK / Atom Editor	Amazon AWS (Maria)	Maria, Raphael & Emil
User Interface	Web	W3C / Atom Editor, Visual Studio (Code), PHP-Storm	Lokaler XAMPP Apache Server (Raphael)	Raphael, Daniela
Dokumentation	API	Deutsch / Google Docs	→ PDF	Daniela, Jeder
Versionskontrolle & Review	Sourcecode	GIT	Github	Emil, restliches Projektteam

Meilensteine

1. Einrichtungen
 - a. Individuelle IDEs
 - b. Github
 - c. AWS
2. Entwicklungen der Anwendungen
3. Deployment der Anwendungen
4. Dokumentation abgeschlossen

Beschreibung der APIs

UI - W3C

Überlegungen & Fortschritt

- Darstellung in einer Web-Oberfläche
 - wir programmieren lieber Webseiten statt Apps
 - funktioniert auf allen Betriebssystemen/Endgeräten
- Seiten: Startseite (inklusive Wetter), Lichter, Jalousien
- W3C - PHP, HTML, CSS, JavaScript

- ☒ Titelseite — Menü
- ☒ Unterseiten
 - ☒ Jalousien
 - ☒ Einzelsteuerung
 - ☒ Mastersteuerung
 - ☒ Windwächter
 - ☒ Lichter
 - ☒ Einzelsteuerung
 - ☒ Mastersteuerung
 - ☒ Szenen
- ☒ Schöne Buttons → Toggle Buttons mit Bildern
- ☒ Flexible Listen
- ☐ Zugriffsschutz (.htaccess?) - OPTIONAL!

Funktion und Programmierung

Hauptseite

- Wetterübersicht
 - Temperatur
 - Windgeschwindigkeit, Richtung
 - Wettervorhersage 7 Tage (Temperatur, Windgeschwindigkeit, Windrichtung)
- Buttons zu Licht- / Jalousien-Unterseiten

```
$soap = new SoapClient(  
    wsdl: null,  
    array(  
        "location" => "http://sedesed.ddns.net:8080/SOAP_API/services/returnMethodes",  
        "uri" => "http://sedesed.ddns.net:8080/SOAP_API/services/returnMethodes",  
        "soap_version" => SOAP_1_1,  
        "trace" => 1  
    )  
);
```

Abbildung: PHP-Klasse für SOAP

Erstellen der Verbindung mit einer PHP-Klasse, die selbstständig die Verbindung aufbauen kann. Dafür wird nur die URL zur Anfrage benötigt. Zwei verschiedene Abfragen sind möglich: Wettervorhersage und aktuelle Wetterdaten.

Als Antwort kommt ein zweidimensionales Array, welches mithilfe von einer Schleife in ein lokales Array abgespeichert wird. Für die aktuellen Daten ist das lokale Array zweidimensional, für die Wettervorhersage dreidimensional.



Abbildung: User Interface - Hauptseite

Unterseite Licht

- Button Home
- Element hinzufügen → POST-Formular
- Darstellung alle Lichter (Tabelle)
 - Icon (0 % aus, 10 - 100 % ein)
 - Prozentwert
 - + / - Buttons (+/- 10 %)
 - Toggle Switch (aus - 0 %, ein - 100 %)
 - Löschen
- Buttons alle Lichter ein/aus
- Buttons für Stimmungen
 - Lesemodus → Ersten zwei auf 70 %, alle anderen Lichter aus
 - Romantikmodus → Abwechselnd 30 % / 0 %
 - Abendmodus → Ersten zwei auf 20 %, alle anderen Lichter aus
 - Gamingmodus → Stellt der Reihe nach die einzelnen Lichter auf 0 %, 20 %, 40 %, 60 %, 80 %

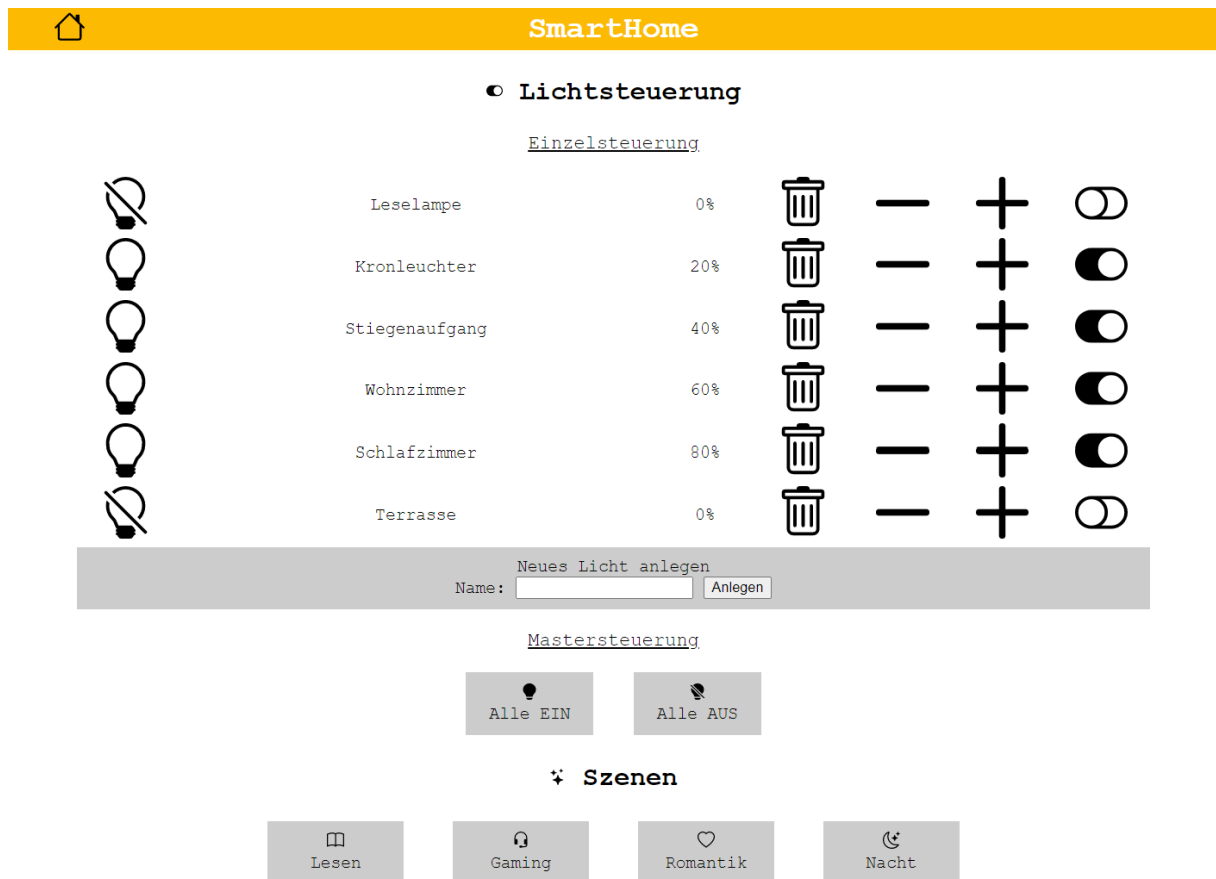



Abbildung: User Interface - Unterseite Lichtsteuerung


























Unterseite Jalousien

- Buttons Home
- Element hinzufügen → POST-Formular
- Darstellung aller Rollläden (Tabelle)
 - Icon (0 % oben, 10 - 100 % unten)
 - Prozentwert
 - Button Pfeile rauf/runter (rauf/runter in 10 %-Schritten)
 - Button Doppelpfeil rauf/runter (komplett rauf wenn zumindest 10 % unten, runter wenn komplett oben)
 - Löschen
- Buttons alle Rollläden rauf/runter
- Aktivitätsanzeige Windwächter


SmartHome



- Jalousiesteuerung

Einzelsteuerung

	Dachfenster	40%				
	Wohnzimmer	70%				
	Wintergarten	0%				
	Westseite	40%				
	Ostseite	0%				

Neue Jalousie anlegen
 Name:

Mastersteuerung

 Alle HOCH
  Alle RUNTER

- Windwächter

Windwächter aktiv: ☒

Abbildung: User Interface - Unterseite Jalousiensteuerung

Lessons Learned

Aktivierung der PHP-SOAP-Klasse: Damit die Verbindung zum SOAP-Server funktioniert muss am XAMPP-Server im File "php.ini" die Zeile "extension=soap" einkommentiert werden, das heißt das Semikolon am Anfang der Zeile muss entfernt werden.

Absenden eines Formulars ohne Weiterleitung auf die Zieladresse:

Problem: Absenden des Formulars leitet auf Zielseite (in action-Tag spezifiziert) weiter - dies ist unerwünscht, es soll keine Weiterleitung stattfinden, jedoch soll der damit verbundene POST-Request trotzdem gesendet werden.

Lösung: Benutzung des target-Tags im Formular, sodass die mit dem POST-Request verbundene Weiterleitung nicht im selben Fenster aufgerufen wird, sondern in einem versteckten (style-Tag: style="display :none") iFrame "angezeigt" (oder eben nicht) angezeigt wird. Somit bekommt der Benutzer davon nichts mit, der POST-Request wird aber trotzdem ausgeführt.

```

<iframe name="dummyframe" id="dummyframe" style="display: none;"></iframe>
<form action="<?php
echo $URL_BASE.$URL_TYPE;
?>" id="newLight" target="dummyframe" method='post'>
  <label for="name">Name: </label>
  <input type="text" name="name" id="name" maxlength="30">
  <button type="submit" onclick="location.reload()">Anlegen</button>
</form>

```

SmartHome Server - REST

Überlegungen

- Python FLASK-Server
- Abfragbare Daten:
 - Daten kommen als simpler String und einzeln
 - Lichter
 - Name, der im Pfad stehen wird
 - Dimmstufe (0 = aus, 10 %-Schritte, 100 = ganz an)
 - Jalousien
 - Name, der im Pfad stehen wird
 - % geschlossen (0 % = ganz offen, 10 %-Schritte, 100 % = ganz zu)
- Zur zumindest zur Ausführungszeit gespeicherten Zustände in Python ein Key-Value-Array mit
 - Spalte 1: Name der Ressource (z. B. /resource/light1)
 - Spalte 2: Wert (z. B. 10 für 10 %)

☒ ~~Speicherung der Zustände in einem temporären globalen 2D-Array~~

☒ ~~Annahme von Requests und Zurückschicken von Strings/Werten~~

☒ ~~Lichter~~

☒ ~~Einzelsteuerung~~

☒ ~~Mastersteuerung~~

☒ ~~Szenen~~

☒ ~~Jalousie~~

☒ ~~Einzelsteuerung~~

☒ ~~Mastersteuerung~~

☒ ~~Windwächter~~

Funktion und Programmierung

- URL: 13.53.174.25:5000/home/light1
 - Verhalten hängt von URL ab
- ~~Benötigte Daten / Zugriff:~~
 - ~~API Key: 12345678909125680~~
- Anfrage:
 - Datenpunkt → Event-Handling
 - Laufzeit-Array mit %-Einträgen
 - RESTful - GET(Abfragen)/POST(Anlegen)/PUT(Verändern)/DEL(Löschen) - einzelne Sachen
 - 1. GET: http://IP/resource/light1 → Zustand kommt als String zurück
 - 2. POST: http://IP/resource/light2?state=0 (wird angelegt) --> Nix / http OK
 - Sonderfall: Wenn es bei POST es schon gibt --> updatet den Wert, falls vorhanden
 - 3. DEL: http://IP/ressource/light1 (wird gelöscht) --> http OK

- 4. PUT: <http://IP/ressource/light2?state=70> ⇒ wir schicken einen String und die Werte werden geupdatet
- Alle Datenpunkte abfragen: → JSON
 - GET: <http://IP/resource/lights/> <http://IP/resources>

REST API functions

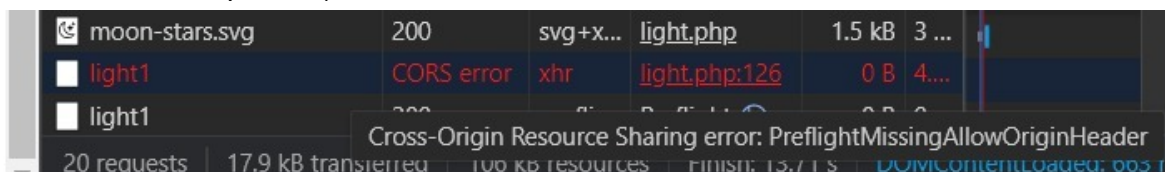
HTTP-Methode & Resource	Parameter	Consumes	Produces	Description
GET: /resource/lights			json	Returns all light items
GET: /resource/shutters			json	Returns all shutter items
GET: /resource/lights/{itemname}	itemname		String	Returns the state of the item
GET: /resource/shutters/{itemname}	itemname		String	Returns the state of the item
GET: /resource/windwatcher			String	Returns 1 if the windwatcher was triggered, otherwise 0
POST: /resource/lights	name	Form data	item	Creates a new light item
POST: /resource/shutters	name	Form data	item	Creates a new shutter item
PUT: /resource/{itemname?state=0}	Itemname? state		New state	Updates the state of the item
PUT: /resource/lights/{itemname?state=0}	Itemname? state		New state	Updates the state of the item

PUT: /resource/shutters/{itemname?state=0}	Itemname? state		New state	Updates the state of the item
DELETE: /resource/lights/{itemname}	itemname		String	Deletes the item
DELETE: /resource/shutters/{itemname}	itemname		String	Deletes the item

Lessons Learned

Fehlende CORS-Messages:

Problem: FLASK-Server sendet per Default keine CORS (Cross-Origin Resource Header)-Informationen mit, Browser akzeptiert dies nicht (siehe Screenshot aus den Browser-Entwickleroptionen)



Untersuchung im Browser: Fehlerrückgabe

Durch installieren der flask-cors library, import cross_origin und das hinzufügen des decorators @cross_origin() zu den Funktionen konnte das Problem gelöst werden.

```
from flask_cors import CORS, cross_origin

@app.route('/resource/lights', methods = ['POST'])
@cross_origin()
def lightX_post():
    item = request.form['name']
```

Python SOAP-Integration schwer:

Die Abfrage des SOAP Wetter-Service hat sich als eher kompliziert herausgestellt, deshalb wird derzeit ein RESTful-Wetter-Service (openWeatherMap) abgefragt, um die aktuellen Winddaten zu bekommen, um bei Starkwind die Jalousien hinaufzufahren.

AWS EC2 Eigenheiten:

Am AWS Linux2 muss beim Installieren von libraries install apt-get yum verwendet werden.

Wetter Service - SOAP

Überlegungen

- Verhältnismäßig simple SOAP-Implementierung per Eclipse JEE & Dynamic Web Service (Tomcat v8.0) möglich.
 - Wetterdaten sollen simuliert werden
 - Es besteht die Option echte Daten aus dem Netz zu laden und dann mit eigener Verarbeitung weiterzugeben.
-
- ☒ ~~Erstellung eines lauffähigen Service~~
 - ☒ ~~Erstellung der notwendigen Methoden~~
 - ☒ ~~Installation des Servers~~
 - ☒ ~~Online Deployment~~
 - ☐ ~~AWS~~
 - ☒ ~~→ Absichtliches privates OnPremise~~
 - ☒ ~~Portmapping~~
 - ☒ ~~Dynamic DNS~~
 - ☐ ~~Erweiterung des Dienstes um~~
 - ☐ ~~API Schlüssel~~
 - ☐ ~~Passwort~~
 - ☐ ~~Echte Abfragen~~

Funktion und Programmierung

- URI: http://sedesed.ddns.net:8080/SOAP_API/services/returnMethodes (privat)
- Aufbau: HTTP-POST / SOAP formatted Document
- Anfrage: Erfolgt per simplem Invoke (z.B. über soapClient in PHP)
- Über SOAP abfragbare Daten:
 - Temperatur
 - Windgeschwindigkeit
 - Windrichtung

```
<soap:envelope>
  <soap:body>
    <myMethod>
      <x>5</x>
      <y>5.0</y>
    </myMethod>
  </soap:body>
</soap:envelope>
```

Funktion & Rückgabetypp	Rückgabedaten	Beschreibung / Methodenweg
String windDirNow()	Windrichtung	Ausgabe nach randomisierter Auswahl aus vorgegebenen String-Array
String windSpeedNow()	Windgeschwindigkeit	Erzeugung des Ausgabe-Strings nach Randomisierung und Rundung

String tempNow()	Temperatur	Erzeugung des Ausgabe-Strings nach Randomisierung und Rundung
String[] weatherDatasetNow()	Einmaliges Array: Temperatur Windgeschwindigkeit Windrichtung	Aufruf der entsprechenden Kernfunktionen; Delimitierung per “.”
String[] weatherDatasetWeek()	Siebenmaliges Array: Temperatur Windgeschwindigkeit Windrichtung	Aufruf der Kernfunktionen; Delimitierung per “.”; Mehrmalige Ausführung;

Anfrage-Antwort-Aufbau

Die **SOAP-Requests** sind wie im folgenden Ausschnitt strukturiert. Anzumerken ist hierbei nur, wie im Body der Funktionsname im Query-Element steht.

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:q0="http://WeatherAPIsoap.sa19b501.bsa4"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <q0:weatherDatasetWeek/>
  </soapenv:Body>
</soapenv:Envelope>
```

Beispiel - SOAP-Request: Windrichtung

Beispielhafte **SOAP-Antworten** sind wie in den folgenden Ausschnitten strukturiert. Hierbei ist ersichtlich, dass aus einer Response der Funktion meist noch ein weiteres Element an sich eingefügt wurde.

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <windDirNowResponse
xmlns="http://WeatherAPIsoap.sa19b501.bsa4">
      <windDirNowReturn>0</windDirNowReturn>
    </windDirNowResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Beispiel - SOAP-Antwort: Windrichtung

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <weatherDatasetWeekResponse xmlns="http://WeatherAPIsoap.sa19b501.bsa4">
      <weatherDatasetWeekReturn>temp:36.68671</weatherDatasetWeekReturn>
      <weatherDatasetWeekReturn>windDir:S</weatherDatasetWeekReturn>
      <weatherDatasetWeekReturn>windSpeed:49.670673</weatherDatasetWeekReturn>
      <weatherDatasetWeekReturn>temp:39.554493</weatherDatasetWeekReturn>
      <weatherDatasetWeekReturn>windDir:NO</weatherDatasetWeekReturn>
      <weatherDatasetWeekReturn>windSpeed:76.94222</weatherDatasetWeekReturn>
      <weatherDatasetWeekReturn>temp:21.436735</weatherDatasetWeekReturn>
      <weatherDatasetWeekReturn>windDir:S</weatherDatasetWeekReturn>
      <weatherDatasetWeekReturn>windSpeed:4.295079</weatherDatasetWeekReturn>
      <weatherDatasetWeekReturn>temp:34.86913</weatherDatasetWeekReturn>
      <weatherDatasetWeekReturn>windDir:OS</weatherDatasetWeekReturn>
      <weatherDatasetWeekReturn>windSpeed:23.450634</weatherDatasetWeekReturn>
      <weatherDatasetWeekReturn>temp:19.907665</weatherDatasetWeekReturn>
      <weatherDatasetWeekReturn>windDir:NW</weatherDatasetWeekReturn>
      <weatherDatasetWeekReturn>windSpeed:2.4901247</weatherDatasetWeekReturn>
      <weatherDatasetWeekReturn>temp:40.49584</weatherDatasetWeekReturn>
      <weatherDatasetWeekReturn>windDir:NW</weatherDatasetWeekReturn>
      <weatherDatasetWeekReturn>windSpeed:4.7264915</weatherDatasetWeekReturn>
      <weatherDatasetWeekReturn>temp:43.937634</weatherDatasetWeekReturn>
      <weatherDatasetWeekReturn>windDir:W</weatherDatasetWeekReturn>
      <weatherDatasetWeekReturn>windSpeed:3.3189797</weatherDatasetWeekReturn>
    </weatherDatasetWeekResponse>
  </soapenv:Body>
</soapenv:Envelope>

```

Beispiel - SOAP-Antwort: Wochen-Datensatz

```

<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <windDirNowResponse
xmlns="http://WeatherAPIsoap.sa19b501.bsa4">
      <windDirNowReturn>0</windDirNowReturn>
    </windDirNowResponse>
  </soapenv:Body>
</soapenv:Envelope>

```

Beispiel - SOAP-Antwort: Error

Lessons Learned

- **Eclipse-JEE verwenden:** Am Anfang schon gab es Probleme mit der Nutzung der Java-IDE. Es waren nicht die richtigen Plugins auffindbar, weshalb ich Eclipse mit JEE-Gesamtpaket komplett neu aufsetzen musste.
- **Tomcat - Nutzen einer älteren Version:** Wie sich aufgrund von Ausführungs-Schwierigkeiten herausgestellt hat, hat sich das Downgrade auf Tomcat v8.0 als gute Idee herausgestellt.
- **Funktionen am Anfang kleinschreiben:** Bei der Integration der Methoden und dem Aufsetzen als Server vermochte es der Tomcat-Server nicht Methoden mit großem Buchstaben am Anfang auszuführen. Der Fix war deshalb entsprechend.
- **Übergabe eines JSON-Dokuments per SOAP ist sinnfrei:** Nachdem die simpleren Rückgaben als String funktionstüchtig waren, waren die komplexen Datenstruktur-Rückgaben eine größere Hürde, die ich per JSON-Dokument zu lösen gedachte. Dies war aufgrund einer Begrenzung der Rückgabelänge nicht möglich und bei einem XML-Zusammenhang als weitere Konsequenz nicht sinnvoll. In dem Sinne wurde auf eine simplere Variante zurückgegriffen, wobei die Rückgabe als String Array stattfindet und die Werte nach einem Delimiter mitgegeben werden.
- **SOAP ist NICHT simpel:** Allein der nötige Programmieraufwand wurde der simpleren Funktion nicht gerecht.
- **Unit-Tests mit eingebauten Tools schwer:** Aufgrund von der Modularisierung und veraltetem Code war es schwer/nicht möglich den Test-Client zum Laufen zu bringen. Wahrscheinlich wird das JEE-SOAP-Paket nicht mehr gewartet.