TKN Telecommunication Networks Group

Technical University Berlin

Telecommunication Networks Group

TWIST Actu: A RESTful Platform for Remote Experimentation With Building Automation Sensors and Actuators

Sunkara Vinodh Kumar, Vlado Handziski

svinodhkumar.1031@gmail.com,handziski@tkn.tu-berlin.de

Berlin, August 2012

TKN Technical Report TKN-12-003

TKN Technical Reports Series

Editor: Prof. Dr.-Ing. Adam Wolisz

Abstract

The **TWIST Actu** platform is aimed at extending the capabilities of the TWIST testbed at TU Berlin with support for remote experiments involving building automation sensors and actuators, as part of a wider effort to migrate the testbed from a pure sensor network testbed to one that can also effectively host more challenging cyber-physical system experimental scenarios. In this demo paper we summarize the main features of the hardware and software architecture of TWIST Actu, focusing on the design of the RESTful remote experimentation API that supports a "Testbed as a Service" model of use of the testbed resources. Finally, we present the prototype implementation of the platform and the scenario for demonstrating its capabilities.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The TKN Wireless Indoor Sensor Network Testbed (TWIST) [5] is a flexible testbed architecture implemented on top of open hardware and software. The TWIST instance at TU Berlin is one of the largest indoor sensor network testbeds with public remote access. It integrates 204 sensor nodes, distributed over three floors of the TKN building, offering more than 1500 m^2 of instrumented office space.

The **TWIST Actu** platform extends the hardware capabilities of the TWIST testbed with typical building automation sensors and actuators like heating plate valve controllers, light switches/dimmers, window tilt and blinds controllers. Wrapped behind an expressive and flexible remote experimentation API, the TWIST Actu extension enables new test scenarios like:

- evaluation of different Internet/HVAC interfacing approaches, middleware, programming abstractions;
- evaluation of interference effects on the correct operation of different sensing/actuation control loops;
- evaluation of the impact of coexistence problems between different radio technologies on the operation of building automation systems, etc.

The selection process for the hardware platform used in the TWIST Actu extension has been conducted following general criteria like maturity, availability, and level of support.

Since the home automation hardware is retrofitted into an "old" building that is already in use, the ease of integration with the existing infrastructure like electrical wiring, heating plates, lights, etc. has also been a major decision factor. Similarly, the need for good indoor RF propagation characteristics and compatibility with the existing sensor node platforms in the testbed has motivated us to seek a platform that operates in the 868 MHz ISM band with support for bidirectional communication between the actuators and the control nodes, enabling evaluation of ARQ protocols.

Following these selection criteria, the TWIST Actu extension has been realized using products from the HomeMatic line of sensors and actuators from the ELV/eQ-3 group [6]. The HomeMatic system is one of the most popular "retrofitting" building automation systems

in Germany. It features a very large product palette and its products are available through major distributors of electronic equipment.

The HomeMatic nodes use a Texas Instruments CC1100 radio in the 868 MHz band, with a custom communication protocol named BidCos. The protocol uses a form of XOR obfuscation and initial ASK-protected handshake, which have already been reverse-engineered. This opens the opportunity to freely communicate with the HomeMatic equipment using any sensor node platform that has a radio compatible with the CC1100, enabling low-level integration with the rest of the TWIST sensor networks infrastructure. In the current configuration, illustrated on Figure 1.1, the TWIST testbed has been extended with seven different types of HomeMatic nodes, installed in the CPS Lab on the second floor of the TKN building.

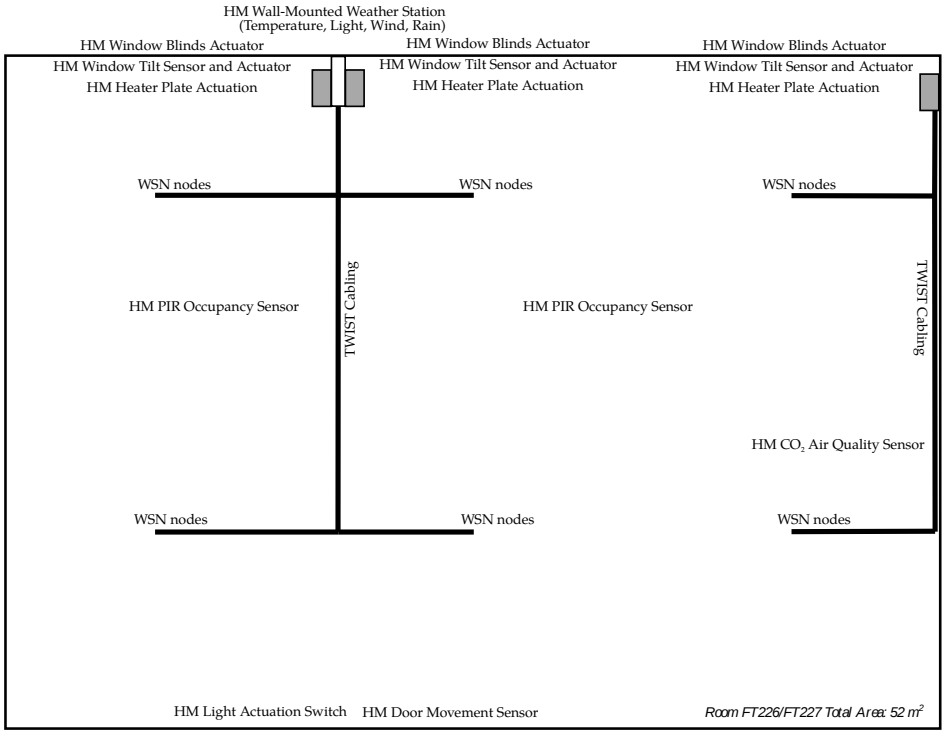

Figure 1.1: Current deployment configuration of the TWIST Actu extension

Chapter 2

TWIST Actu Remote Experimentation API

The user-facing service API for remote experimentation with building automation actuators has been designed as an extension of the COTEFE Testbed Abstraction API (TAA) [2]. Following the Representational State Transfer (REST) architectural style [4], the API is specified as a set of resources which are manipulated through a uniform method set, comprised of the standard HTTP methods like GET, PUT, DELETE, POST, etc. The role and the semantics of the methods are independent from the resources that are being manipulated, specified by a Universal Resource Locator (URL), as shown on Figure 2.1.

	GET	PUT	DELETE	POST
Resource URL Eg: http://cotefe.net /nodes/12	Retrieve	Create/ Replace	Delete	Append/ Modify
	Safe Idempotent Cacheable	Idempotent	Idempotent	Not Safe Not Idempotent

Figure 2.1: Resource manipulation using a uniform method set in REST

The abstraction of the access to the building automation sensors and actuators in TWIST Actu has been achieved through two new resources that extend the resource model of the TAA: *Channel* and *Parameter*.

The API supports two types of channel resources: sensors and actuators. The sensor channels perform a supervisory function. They sense a particular parameter in the environment and generate events when necessary. Actuator channels perform control functions and influence the state of physical objects in the environment. Both sensor and actuator channels are characterized by state values, data points and attributes which are exposed through the parameter resource concept (Table 2.1).

The representation of the resources is realized using JSON serialization that simplifies the development of JavaScript client implementations and increases the human-readability and

RESOURCES	ANALOGY(with real devices)	ATTRIBUTES	FIELDS
Node	Parent Device	id	Char
		native_id	Integer
		name	Char
		platform	ForeignKey
		location_x	Decimal
		location_y	Decimal
		location_z	Decimal
		datetime_created	DateTime
		datetime_modified	DateTime
Channel	Channels in a parent device	id	Char
		name	Char
		node	ForeignKey
		is_sensor	Boolean
		is_actuator	Boolean
Parameter	Data Points for a channel	id	Char
		channel	ForeignKey
		name	Char
		value	Float
		type	Char
		unit	Char
		min	Float
		max	Float

Table 2.1: Extended TAA resources for the TWIST Actu services

encoding efficiency in comparison to XML-based solutions.

Figure 2.2 documents the associated URL hierarchy. Starting from the top, the actuator devices in the TWIST Actu extension are considered to be normal System Under Test (SUT) TWIST nodes. For example, issuing a HTTP GET command to the URL:

- `https://www.twist.tu-berlin.de:8001/nodes`

returns a container representation of all nodes that are currently accessible on TWIST. These nodes include both the new Homematic actuator nodes and the existing TWIST sensor nodes:

```
[
{
"id": "GEQ0207622",
 "uri":"https://www.twist.tu-berlin.de:8001/nodes/GEQ0207622",
 "media_type": "application/json",
 "name": "HM-CC-SCD"
},
{"id": "HEQ0138342",
 "uri":"https://www.twist.tu-berlin.de:8001/nodes/HEQ0138342",
 "media_type": "application/json",
```

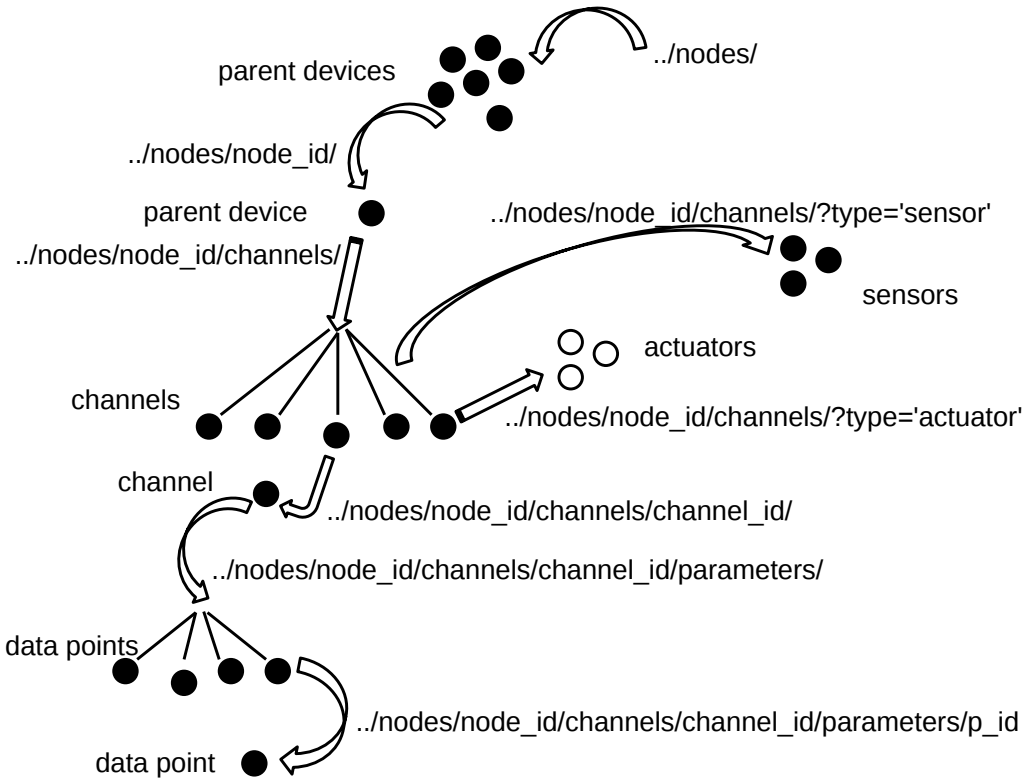

Figure 2.2: TWIST Actu resource hierarchy

```
 "name":"HM-CC-VD"
},
]
```

Discovery of nodes from a particular platform can be performed using the platform filter. For example, to receive a container representation of all HomeMatic nodes currently accessible on TWIST, one needs to issue a HTTP GET command to the URL:

- `https://www.twist.tu-berlin.de:8001/nodes/?platform=homematic`

Like other SUT nodes, each actuator device is uniquely identified through a serial number which is leveraged for construction of its URL. For example, a HTTP GET on the URL:

- `https://www.twist.tu-berlin.de:8001/nodes/IEQ0042249`

provides a detailed description of a HomeMatic actuator with serial number "IEQ0042249":

```
{
"id": "IEQ0042249",
"uri": "https://www.twist.tu-berlin.de:8001/nodes/IEQ0042249",
"media_type": "application/json",
"name": "HM-Sec-Win",
```

```
"native_id": "null",
"platform":
{
"id": "homematic",
"uri": "https://api.cotefe.net/platforms/homematic",
"media_type": "application/json",
"name": "HomeMatic"
},
"channels":
[],
"location_x": 0,
"location_y": 0,
"location_z": 0,
"datetime_created": "2012-07-09T11:24:06",
"datetime_modified": "2012-07-09T11:24:06"
}
```


The list of all channels supported by this actuator can be retrieved by issuing the HTTP GET method on the URL:

- `https://www.twist.tu-berlin.de:8001/nodes/IEQ0042249/channels`

whereas the same method issued against the URL:

- `https://www.twist.tu-berlin.de:8001/nodes/IEQ0042249/channels/1`

provides a detailed representation of the HomeMatic channel with address "IEQ0042249:1":


```
[
{
"id": 0,
"uri": "https://www.twist.tu-berlin.de:8001/nodes/IEQ0042249/channels/0",
"media_type": "application/json",
"name": "MAINTENANCE"
},
{
"id": 1,
"uri": "https://www.twist.tu-berlin.de:8001/nodes/IEQ0042249/channels/1",
"media_type": "application/json",
"name": "WINMATIC"
},
{
"id": 2,
"uri": "https://www.twist.tu-berlin.de:8001/nodes/IEQ0042249/channels/2",
"media_type": "application/json",
"name": "AKKU"
}
]
```


The data points in the internal HomeMatic APIs which encode the internal state of a particular channel are exposed through the parameter resource. The list of all data points

for a particular channel is thus obtained by the issuing a HTTP GET request against a URL in the form:

- `https://www.twist.tu-berlin.de:8001/nodes/IEQ0042249/channels/1/parameters`

  whereas a HTTP GET request against the URL:

- `https://www.twist.tu-berlin.de:8001/nodes/IEQ0042249/channels/1/parameters/2`

  provides a detailed representation of a particular parameter:


```
{
"id": 2,
"uri": "https://www.twist.tu-berlin.de:8001/nodes/IEQ0042249/channels/1/parameters/2",
"media_type": "application/json",
"name": "LEVEL",
"type": "FLOAT",
"value": 0,
"unit": "100%",
"min": 0,
"max": 1,
"channel":
{
"id": 1,
"uri": "https://www.twist.tu-berlin.de:8001/nodes/IEQ0042249/channels/1",
"media_type": "application/json",
"name": "WINMATIC"
},
"node":
{
"id": "IEQ0042249",
"uri": "https://www.twist.tu-berlin.de:8001/nodes/IEQ0042249",
"media_type": "application/json",
"name": "HM-Sec-Win"
},
"datetime_created": "2012-07-09T11:24:07",
"datetime_modified": "2012-07-09T11:31:38"
}
```

Chapter 3

TWIST Actu Prototype

We have developed a prototypical implementation of the TWIST Actu API which has been deployed for testing. Figure 3.1 illustrates the system architecture of the current prototype. It shows how the XMLRPC-based interfaces offered by the HomeMatic Central Control Unit (CCU) have been integrated and abstracted via an extended CONET Testbed Adaptation Server implemented with Django [7].

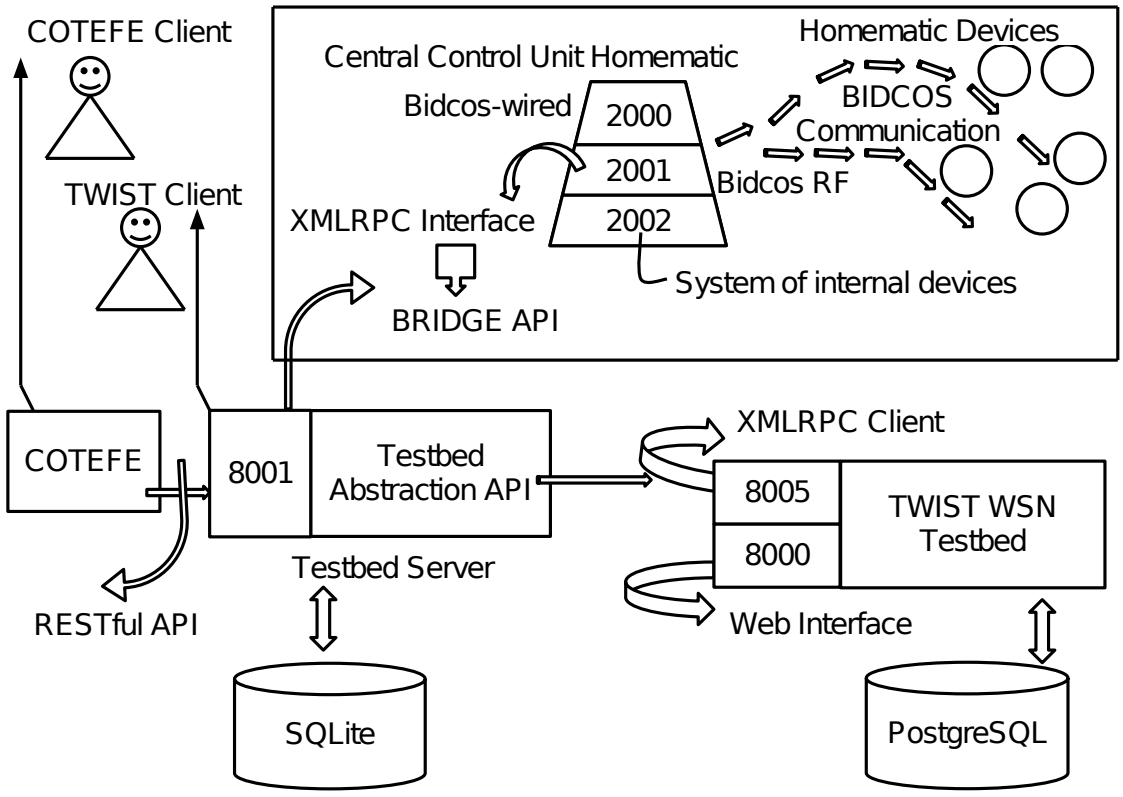

Figure 3.1: System architecture of the TWIST Actu extension

To simplify the adaptation of this RPC-centric API to the user-facing REST API, we

have leveraged an intermediate gateway adaptation component called "Homematic Bridge" with the following API signature:

API for Winmatic HM-Sec-Win

- openWindow(self, request, address)
- closeWindow(self, request, address)
- tiltWindow(self, request, address, value)
- getWindowTilt(self, request, address)

API for Channel 1 of HM-CC-TC

- getTemperature(self, request, address)
- getHumidity(self, request, address)

API for Channel 2 of HM-CC-TC

- setClimateRegulator(self, request, address, value)
- getClimateRegulator(self, request, address)

API for CO2 sensor HM-CC-SCD

- getCO2(self, request, address)

API for Radio Control Valve (Heater) HM-CC-VD

- getHeaterState(self, request, address)

API for motion detector HM-Sec-MDIR

- getBrightness(self, request, address):
- getMotion(self, request, address)
- getNextTransmission(self, request, address)
- setNextTransmission(self, request, address, value)

The Homematic CCU provides the facility of sending notifications about important events reported by the sensor and actuator devices. To this end, the CCU acts as a XMLRPC client that can "push" notifications to a remote XMLRPC server. Combined with the "direct path" where the CCU acts as a server, this creates a reverse XMLRPC bridge enabling command and notification flow in both directions.

To adapt the RPC-based notification mechanism to the requirements of the TWIST Actu experimentation API, the notifications are offered to the testbed user using the HTML5 WebSocket protocol [3].

The WebSocket mechanism enables much more efficient "pushing" of notification from the sensors and actuators than the traditional methods of HTTP-polling or HTTP-streaming. Figure 3.2 illustrates the differences in the interaction patterns between these two methods, and the benefits of the WebSocket approach (Figure 3.3).

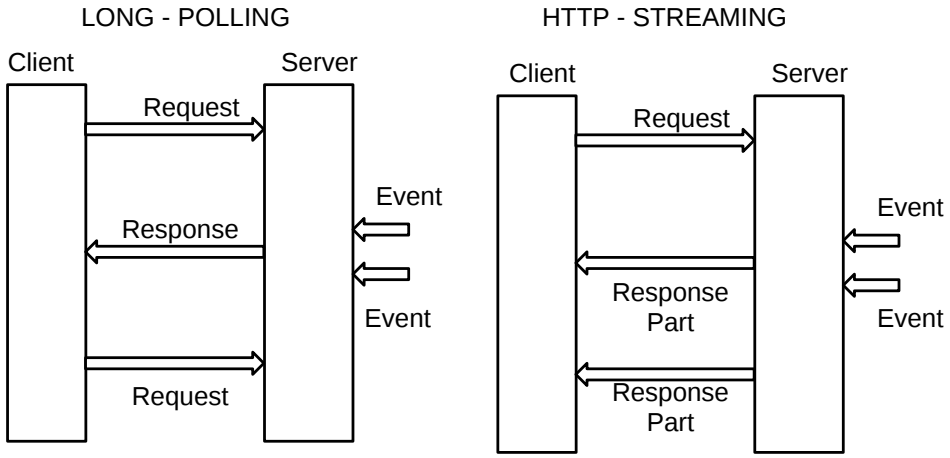

Figure 3.2: Client–Server interaction patterns in HTTP long polling and streaming

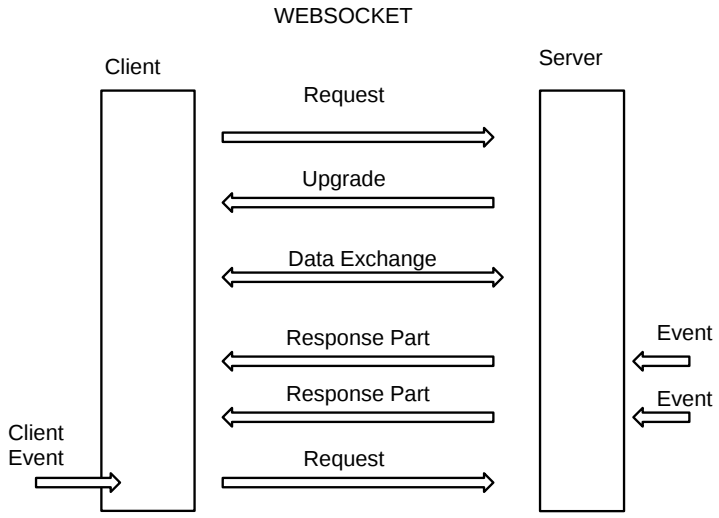

Figure 3.3: Client–Server interaction patterns in WebSockets

The WebSocket protocol provides a bidirectional communication service between a web client and a server. After an initial HTTP-based negotiation phase, the connection is "upgraded" to a bi-directional data flow conduit over the same underlying TCP/IP connection between the client and the server (Figure 3.4).

```
{
"Request URL" : "ws://192.168.100.6:9036/",
"Request Method" : "GET",
"Status Code" : "101 Switching Protocols Request Headers",
"Connection" : "Upgrade",
"Host" : "192.168.100.6:9036",
"Origin" : "null",
"Sec-WebSocket-Extensions" : "x-webkit-deflate-frame",
"Sec-WebSocket-Key" : "882suVIB7Cv7+N1vK41nmg==",
"Sec-WebSocket-Version" : 13,
"Upgrade" : "websocket",
"(Key3)" : "00:00:00:00:00:00:00:00",
"Connection" : "Upgrade",
"Sec-WebSocket-Accept" : "+oMgYFzTuhdbN9PThuhLA14B6u8=",
"Server" : "AutobahnPython/0.5.5",
"Upgrade" : "WebSocket",
"(Challenge Response)" :"00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00"
}
```

Figure 3.4: Client–Server negotiation for upgrading a HTTP connection to a WebSocket connection

The WebSocket server in the TWIST Actu prototype has been implemented using the "Autobahn" library [1]. The architecture of the reverse XMLRPC bridge implementation is shown in Figure 3.5.

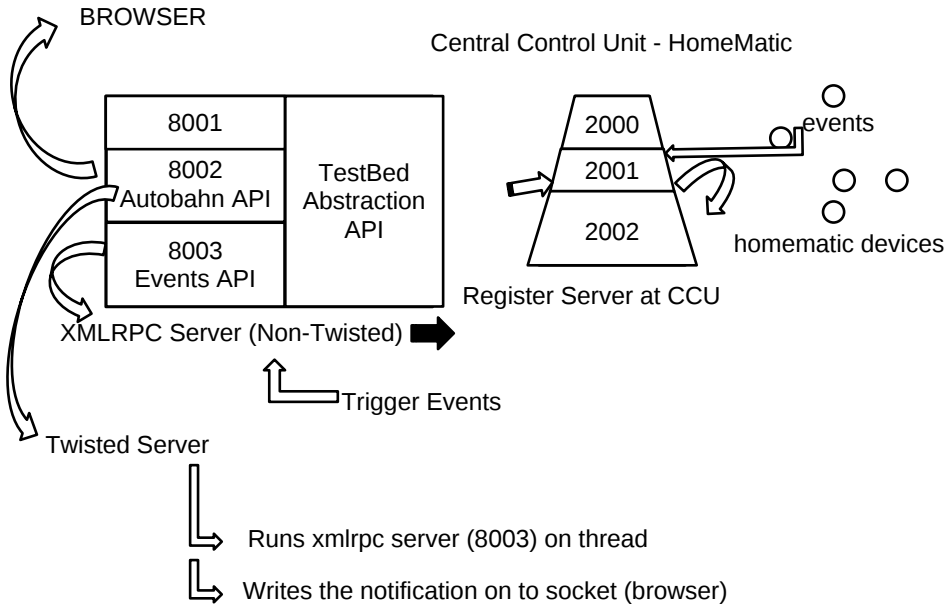

Figure 3.5: Notification "push" using a reverse XMLRPC bridge

Figure 3.6 provides an illustration of the notification push service in TWIST Actu. A

change in the tilt level of the window actuator “IEQ0042251:1” to value “0.245”, triggers a notification encoded in JSON format that is efficiently pushed to the testbed client.

```
{"RESPONSE":
{
"id" : "rf",
"channel" : "IEQ0042251:1",
"parameter" : "LEVEL",
"value" : "0.245"
}
}
```

Figure 3.6: WebSocket-based notification for actuator state change

Chapter 4

Performance Evaluation

To evaluate the performance of the implemented prototype, we have measured the time overhead in the execution of several core operations of the platform. The results are summarized in the form of boxplots, showing the median, lower and upper quartiles, and outliers for one hundred repetitions of the respective operation. Figure 4.1 shows the performance results for the database-related operations. The results for the CCU-related operations are depicted on Figure 4.2. In the following we provide brief description of the evaluated operations and

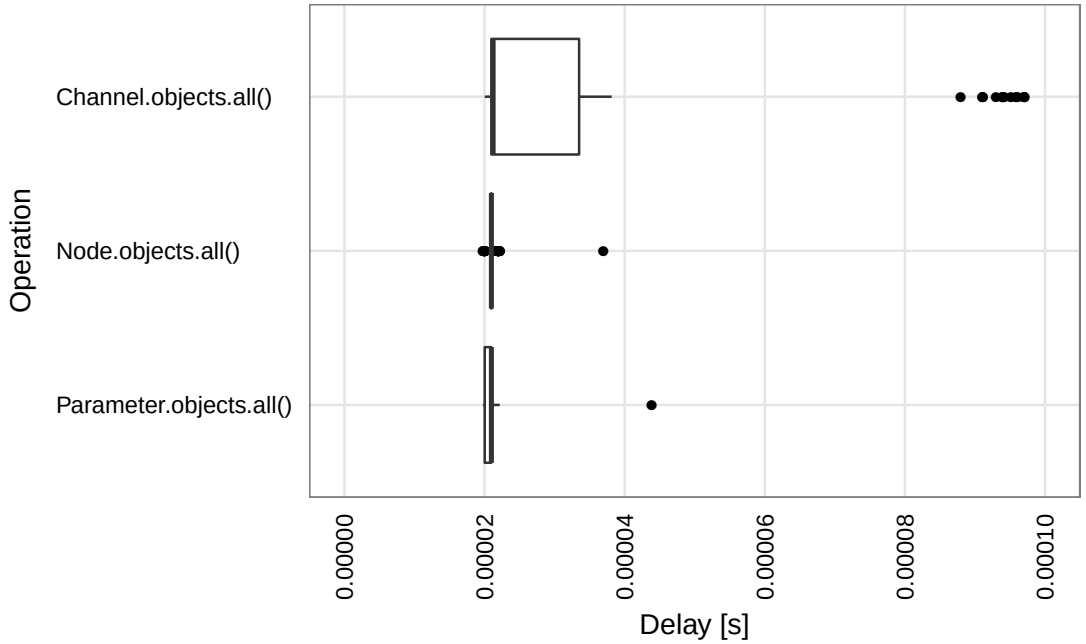

Figure 4.1: Time overhead in the execution of several core database-related operations

Channels from Database The presented boxplot summarizes the time overhead in the execution of the `Channel.objects.all()` operation, measured using the "timeit" module in

Python. The evaluation shows that the average time overhead for channel retrieval from the database is on he order of 0.000035 s. The operation returns a channels list containing all channels associated with all nodes, leading to a relatively larger time overhead compared to other database operations. The channel list in the database is a cached version of the channel data returned by the `proxy.getDevices()` command on the CCU described below.

Nodes from Database The boxplot summarizes the time overhead in the execution of the `Node.objects.all()` operation. The evaluation shows that the average time overhead for node retrieval from the database is on he order of 0.000021 s.

Parameters List from the Database The boxplot shows the time overhead in obtaining the list of Parameters associated with a given channel from the database using the `Parameter.objects.all()` command. The average measured time overhead for the operation is 0.000021 s.

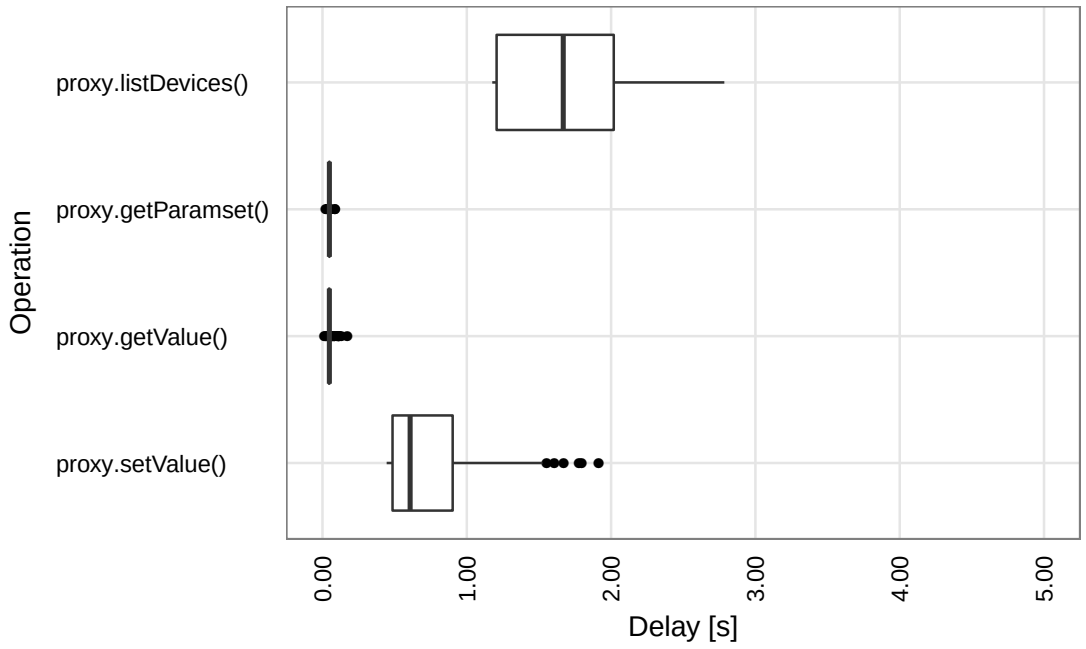

Figure 4.2: Time overhead in the execution of several core CCU-related operations

Devices List from the CCU This operation returns a list of all nodes and channels registered at the CCU. The boxplot summarizes one hundred iterations of the `proxy.listDevices()` command where "proxy" is a ServerProxy instance that manages the communication with the XMLRPC server at the CCU. The evaluation shows that the average time overhead for the retrieval of the devices list from CCU is on the order of 1.67 s.

Parameter Set from the CCU The parameter set from the CCU contains all parameters associated with a given channel. The boxplot depicts the time overhead in the execution of the `'proxy.getParamset()` operation. The average time taken for the parameter set retrieval from the CCU is 0.05 s.

Get Value from the CCU The plot covers one hundred iterations of `proxy.getValue()` command. The returned result represents the value of a particular parameter obtained through the CCU.The obtained average time overhead measure is on the order of 0.05 s.

Set Value at the CCU The `proxy.setValue()` command supports setting a given parameter value at the CCU. As it can be seen on the plot, the average time overhead for updating a parameter value at the CCU is expectantly significantly higher than the simple `proxy.getValue()` retrieval, and is on the order of 0.75 s.

Chapter 5

Acknowledgments

The work has been performed during the first author's internship at the TKN group at TU Berlin. It has been partially supported by EIT ICT Labs, as part of the activity 12149, "From WSN Testbeds to CPS Testbeds".

Bibliography

[1] Autobahn web sockets. `http://autobahn.ws`.

[2] Claudio Donzelli, Vlado Handziski, and Adam Wolisz. Demo Abstract: Testbed-independent experiment specification and execution using the COTEFE platform. In *Proc. of 9th European Conference on Wireless Sensor Networks*, EWSN 2012, February 2012.

[3] I. Fette and A. Melnikov. The WebSocket protocol. Internet Network Working Group RFC6455, 2011.

[4] Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. In *Proc. of the 22nd intl. conference on Software engineering*, ICSE '00, 2000.

[5] Vlado Handziski, Andreas Köpke, Andreas Willig, and Adam Wolisz. TWIST: a scalable and reconfigurable testbed for wireless indoor experiments with sensor networks. In *Proc. of the 2nd Intl. Workshop on Multi-Hop Ad Hoc Networks*, REALMAN '06, May 2006.

[6] HomeMatic platform. `http://www.homematic.com`.

[7] Django, the web framework for perfectionists with deadlines. `http://www.djangoproject.com`.