A Novel Application of CWMP: An Operator-grade Management Platform for IoT

Martin Stusek, Pavel Masek, Krystof Zeman, Jiri Pokorny, Dominik Kovac, Petr Cika, and Franz Kröpfl

***Abstract*—The aggressive expansion of emerging smart devices connected to the Internet infrastructure is nowadays considered as one of the most challenging components of the Internet of Things (IoT) vision. As a particular segment of IoT, the smart home gateways, also named Machine-Type Communication Gateway (MTCG), become an important direction for industry including telecommunication operators. In most cases, the MTCG acts as a bridge between connected smart objects and the public network (Internet). As a consequence of the IoT domain expansion, the separate configuration of each individual Machine-to-Machine (M2M) device is not feasible anymore due to steadily growing numbers of M2M nodes. To perform this task, several novel technologies have recently been introduced. However, legacy protocols and mechanisms for remote network management still retain a certain application potential for IoT. Accordingly, we have investigated the protocol TR-069 with a particular focus on its usability for MTCG. To this end, the software module (bundle) based on the TR-069 for remote configuration and management of MTCG, as well as for controlling the end smart devices, has been developed. We believe that our implementation (available as open source on GitHub) can serve as an important building block for efficient management of future IoT devices. Therefore, TR-069 protocol constitutes a proven and standardized technology and could be easily deployed by most of the network and service providers today. Authors would like to recall that this paper represents extended version of their previously published work at TSP 2016 conference.**

***Keywords*—M2M, MTCG, OSGi framework, TR-069, Remote management**

I. Introduction

Today, Internet of Things (IoT) offers efficient means for interconnection of highly heterogeneous entities and networks, thus bringing a variety of communication patterns, including Human-to-Human (H2H), Human-to-Machine (H2M), and Machine-to-Machine (M2M) communications. IoT in general empowers the industry to develop new technology in unprecedentedly large numbers. New findings from the leading telecommunication players, such as Juniper [1] and Cisco [2], reveal that global retail revenue from smart wearable devices (as one of the IoT segments) will triple by 2016, therefore reaching $53.2 billion by 2019, as compared to the $4.5 billion at the end of 2015. The market over the following five years is expected to be substantially driven by the sales of smart devices, named MTCD (Machine-type Communication Devices) – an important component of this group is represented by smart home gateways, also known as MTCG (Machine-type Communication Gateway) [3], [4].

Presently, the MTCGs become more intelligent and provide new functions for smart data collection and visualization on end-user interfaces. In the light of the recent development in the IoT domain, the MTCG is capable of offering much more than conventional local networking features inside residential buildings [5]. Many devices acting as MTCD, that is, based on different communication technologies (IEEE 802.15.1, 6LoWPAN, ZigBee, Wireless M-BUS, etc.), are currently employing MTCG as an aggregation node providing access to the public network (Internet) [6], [7]. Inspired by these developments, we have recently introduced the concept of multi-purpose Smart Home Gateway (SH-GW) within our outgoing project under the title SyMPHOnY [8].

In this work, we aim at enabling remote configuration for devices in the role of SH-GWs by continuing our line of research [9]. Despite the fact that IoT is changing the conventional communication paradigm in many ways [10], some principles are remaining unchanged; therefore, many legacy technologies can be applied to IoT as well. Following this thinking, we have been investigating the protocol TR-069, well-known by network operators to maintain the Customer Premises Equipment (CPE), as a promising candidate for remote configuration of the IoT nodes, see Fig.1 and Fig.2 where the number of installed TR-069-enabled CPE is shown – with respect to device type and region. To this end, we have developed a SH-GW demonstrator, where the TR-069 is implemented as an extension of OSGi frameworks which are commonly used as the primary middleware layer for smart home gateways shipped by telecommunication operators. In other words, by using the TR-069 on MTCGs, service providers and telecom operators are able to manage and control not only the gateway but also the devices behind (e.g., energy meters, motion sensors, etc.) [11]. This important use case raises many research questions related to the configuration of various devices sets, the remote access capabilities, as well as the choice of cryptographic mechanisms used for data transmission. In this extended version of our previous work [9], we have focused our attention to address most of these issues. Namely, the description of (i) procedures and requirements, and (ii) application logic while using the TR-069 protocol in created setup is given in detail.

The rest of this paper is organized as follows. Section II is devoted to describing the operation principles of TR-069 protocol. Further, in Section III, a detailed description of our developed software implementation for OSGi frameworks together with a practical scenario accounting for all mentioned issues are offered. Finally, the lessons learned during our system development are summarized in the concluding

Manuscript received September 17, 2016; revised December 21, 2016.

M. Stusek, P. Masek, K. Zeman, J. Pokorny, D. Kovac, and P. Cika are with Department of Telecommunications, Brno University of Technology, Brno, Czech Republic (e-mails: martinstusek@phd.feec.vutbr.cz, masekpavel@feec.vutbr.cz, krystof.zeman@phd.feec.vutbr.cz, pokorny.jiri@phd.feec.vutbr.cz, kovacd@feec.vutbr.cz, cika@feec.vutbr.cz).

F. Kröpfl is with Telekom Austria Group, Austria (e-mail: Franz.Kroepfl@telekomaustria.com).

The described research was supported by the National Sustainability Program under grant LO1401. For the research, infrastructure of the SIX Center was used. Authors would like to thank to Telekom Austria Group for access to SIP infrastructure and insight into M2M and its real-life usage.

Section IV.

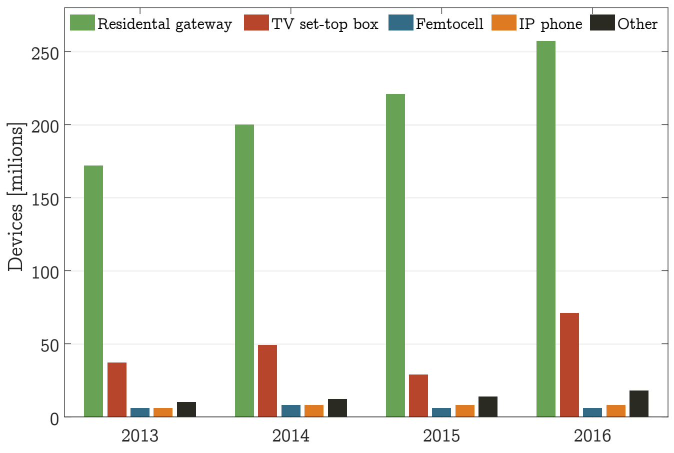


Fig. 1: Number of installed TR-069-enabled CPE by type.

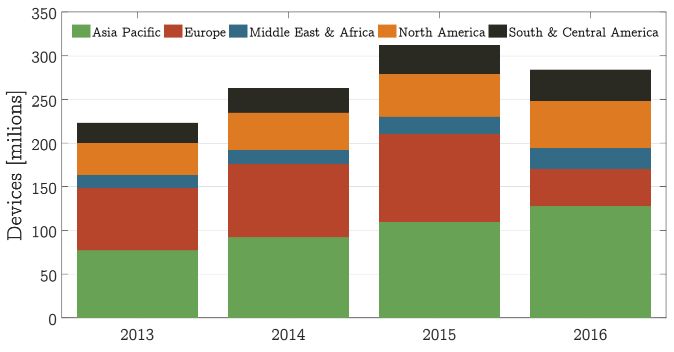


Fig. 2: Number of installed TR-069-enabled CPE by region.

II. Remote Network Configuration Using TR-069

As mentioned in the introduction, the need for remote configuration and management of network nodes brings new challenges to the IoT domain. Fueled by large numbers of M2M devices, the service providers require to control all of the devices in efficient and centralized way. For this purpose, several application layer protocols for remote management of end-user devices have already been introduced by different working groups and standardization bodies [12]. As a well-known and widely used representative, the TR-069 protocol is often utilized by telecom operators [3]. In this section, the functional architecture blocks of TR-069 are described with the emphasis on future implementation as a bundle in OSGi framework.

A. *Protocol Architecture*

TR-069 represents a protocol for encrypted remote self-configuration of CPE from the side of ACS (Auto-Configuration Server). The overall architecture of TR-069 ecosystem is depicted in Fig. 3.

The key element of TR-069 protocol is ACS server that provides information to one or more CPE according to a number of criteria. This mechanism allows for offering a default set of parameters and, furthermore, introduces a possibility of adding

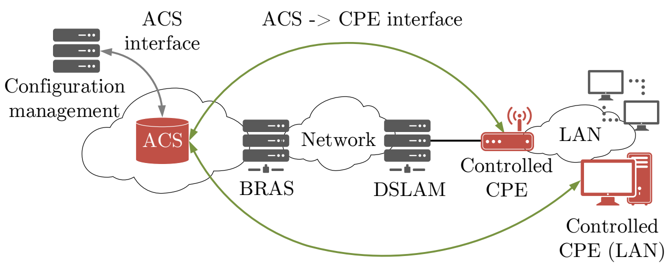


Fig. 3: Architecture of TR-069 ecosystem

new features according to the manufacturer's requirements. Parameters of the connected CPEs are available during the initial connection setup as well as the regular transmission as requests (e.g., providing information about CPE from ACS based on asynchronous, server-initialized[1] connection).

TABLE I: Protocol Layer Summary [12]

Protocol	Description
CPE/ACS Application	The application uses the CPE WAN management protocol for the CPE and ACS, respectively. It is defined locally but is not a part of the CPE WAN.
RPC Methods	The specific RPC methods are defined by the CPE WAN Management Protocol. This includes the definition of the CPE parameters accessible by the ACS using the parameter-related RPC methods.
SSL/TLS	Standard Internet transport layer security protocols – SSL 3.0 or TLS 1.0 are used.
SOAP	A standard XML-based syntax is used to encode remote procedure calls via the SOAP 1.1 protocol.
HTTP	Standard HTTP 1.1.
TCP/IP	Standard TCP/IP.

One of the most important tasks for remote configuration is to allow secure communication for sensitive data like e.g., encryption keys. TR-069 provides tools to download new software / firmware from the ACS server using digital signatures – to verify the integrity of downloaded files at the side of CPE [13]. Further, TR-069 defines a set of parameters that can be used for connection / service diagnostics [12].

1) Protocol Components: The TR-069 protocol architecture includes several unique components comparing to other *dedicated* IoT management protocols (e.g., the RPC (Remote Procedures Calling), see Section II-A3). In addition, TR-069 uses standard protocols, such as SOAP (Simple Object Access Protocol), HTTP (Hypertext Transfer Protocol), SSL / TLS (Secure Sockets Layer / Transport Layer Security), and TCP / IP (Transmission Control Protocol / Internet Protocol) [12]. The overview of complementary protocols acting on different layers is given in Table I.

On top of the supported protocols, TR-069 defines several device types, where each device may be described by a data model containing information about the parameters and

[1]In TR-069 terminology, the connection is called server-initialized, even though the communication is started at the CPE side. This is due to the fact that there is a need for appropriate connection setup of the CPE devices residing in local network where Network Address Translation (NAT) is used.

provided functions for a selected device. Supported TR-069 data models are shown in Table II (highlighted rows stand for the data models implemented in this work).

TABLE II: Data Models [14]

Data model	Compliant device
TR-064 and TR-133	LAN CPE devices
TR-068 and TR-124	Gateway modems
TR-098	Internet gateway device data model for TR-069
TR-104	Provisioning parameters for VoIP CPE
TR-106	Data model template for TR-069-enabled devices
TR-110	Reference model for VoIP configuration
TR-111	Applying TR-069 to remote management of home networking
TR-122	ATA devices
TR-126	Triple-Play QoE (Quality of Experience) requirements
TR-128 and WT-123	TR-069 testing support
TR-131	ACS Northbound interface requirements
TR-135	Data model for a TR-069 enabled STB
TR 140	TR-069 data model for storage service-enabled devices
TR-142	Framework for TR-069-enabled Passive Optical Network (PON) devices
TR-143	Enabling network throughput performance tests and statistical monitoring
TR-157	Component objects for CWMP (UPnP/DLNA device support)
TR-181	Device data model for TR-069
TR-196	Femto access point service data model

2) Security Mechanisms: The TR-069 protocol is designed to ensure the adequate level of security. Therefore, it includes methods for protection against manipulation during the transactions between the ACS server and the end-device (CPE). Further, the security algorithms using multiple levels of authentication are implemented by means of SSL/TLS for communication between the ACS and the CPE [12].

3) Architectural Components: The RPC defines a list of parameters and methods that have to be included at the end-device (CPE) in order to construct and send the TR-069 requests. In the following text, a summary of the most important components is given [12]:

- **Parameters** – RPC method specification defines a generic mechanism allowing the ACS server to read or write parameters for the CPE configuration, and to monitor CPE status and statistics. Each parameter has a name-value structure. The name identifies a particular parameter and has a hierarchical structure similar to the conventional directory listing ones (each level is separated by "." (dot)). The value of a parameter may be one of several defined data types.
- **File Transfers** – In TR-069, the mechanism enabling file download or (optionally) upload is implemented in order to perform tasks, e.g., CPE firmware upgrade or download of vendor-specific configuration files. When the session between ACS and CPE is initiated, the data transmission is performed utilizing HTTP or (preferably) HTTPS. Other protocols, including FTP and TFTP, are supported as well, but used less frequently.
- **CPE Connection Notifications** – TR-069 defines a mechanism allowing CPE to notify the corresponding ACS about various conditions – to ensure that the frequency of CPE-ACS communication remains optimal.
- **Asynchronous ACS-Initiated Notifications** – An important aspect of auto-configuration service is the ability of the ACS server to notify the remote CPE about configuration changes asynchronously. It allows the auto-configuration mechanism to be utilized for services requiring real-time management of the CPE.

4) Procedures and requirements: Protocol TR-069 defines required procedures which are necessary for (i) finding ACS, (ii) establishing a connection or (iii) creating of a session.

- **Finding ACS** – Following methods can be used to get the address of the server:
  - (i) Using DNS (Domain Name System) to get the IP address of the server from URL (Uniform Resource Locator) which is located in CPE.
  - (ii) It is possible to use the part of IP model for automatic configuration, that means using DHCP (Dynamic Host Configuration Protocol) whose messages contain the URL of the server. After that, using the DNS gets the CPE IP address of the server.
  - (iii) CPE can contain a default ACS address which is used if no other URL is defined.
- **Establishing connection** – After successfully getting the address, the connection can be established by CPE or ACS. If NAT (Network Address Translation) is used, connection can be established only from CPE side.
  - Establishing connection from CPE: End device can establish connection anytime it knows the ACS address but it is mandatory to establish connection in these cases:
    - First connection of CPE into the network and first configuration.
    - Device startup.
    - After an interval defined in settings.
    - When a method requires it.
    - Change of ACS address.
    - After any change of parameters.

    To protect the auto-configuration server from congestion, every CPE has a defined maximum value of messages informing the server of parameter changes.
  - Establishing connection from ACS: In this case it can be accomplished with mechanism of announcement of connection request. This parameter is mandatory on CPE side and recommended on ACS side. Establishing a connection from side of the server is possible only when the CPE address is public. In other case only the client can establish a connection.
- **RPC message requirements** – List of methods which are defined in the layer of remote call is shown in Table III.
- **Session management** – All sessions must start with an information message from CPE that contains an initialization HTTP message. That serves for setting up com-

TABLE III: RPC message requirements

Method name	CPE side	ACS side
CPE methods	Answers	Call
GetRPCMethods	Mandatory	Optional
SetParameterValues	Mandatory	Mandatory
GetParameterValues	Mandatory	Mandatory
GetParameterNames	Mandatory	Mandatory
SetParameterAttributes	Mandatory	Optional
GetParameterAttributes	Mandatory	Optional
AddObject	Mandatory	Optional
DeleteObject	Mandatory	Optional
Reboot	Mandatory	Optional
Download	Mandatory	Mandatory
Upload	Optional	Optional
FactoryReset	Optional	Optional
GetQueuedTransfers	Optional	Optional
ScheduleInform	Optional	Optional
SetVouchers	Optional	Optional
GetOptions	Optional	Optional
Server methods	Call	Answers
GetRPCMethods	Optional	Mandatory
Inform	Mandatory	Mandatory
TransferComplete	Mandatory	Mandatory
RequestDownload	Optional	Optional
Kicked	Mandatory	Optional

munication limits and coding at client's side. A session is terminated if server or client do not have to send any more requests or answers to a request. There can be only one existing session at a time. Both, ACS and CPE must handle session (initialization, incoming/outgoing requests, session termination) according to TR-069 standard. In text below set of basic session commitments required by TR-069 is given.

- CPE - Session Initiation: CPE can initialize a session only in case parameters available through its interface are locked. The reason is to protect parameters from change from a different source. This lock can remain active until the end of the session.
- CPE - Incoming requests: End device responds to requests in order in which they have been received. To prevent a deadlock, CPE does not wait for confirmation of previous request from server before it sends another request.
- CPE - Outgoing requests: When CPE has some requests for the server, it may send them in any order with respect to responses being sent from CPE to ACS.
- CPE - Session Termination: Client has to terminate a session when the following conditions are met:
  1) Server has no other requests.
  2) CPE has no other requests.
  3) Client received all unfinished requests from the server.
  4) Client sent all requests to ACS.

  CPE must terminate a session if it has not received any answer from the server in more than 30 seconds. If these conditions are not met, client must continue the session. In case of unexpected session termination, session initiation must start from the beginning.
- ACS - Session Initiation: After receiving the initial Inform request from CPE, the server must respond with the Inform response.
- ACS - Outgoing requests: When the server has requests to send, it may send them with respect to responses sent by ACS to CPE. If the ACS has more than one request to send or there are responses left to send, the ACS must send at least one request or response to CPE. Empty HTTP response is only allowed if ACS has no more requests or no responses to send.
- ACS - Session Termination: If the connection has been established by CPE, then it is also responsible for the session termination. ACS may consider to terminate the session when all the following conditions are met:
  1) CPE has no more requests.
  2) ACS has no more requests.
  3) CPE has sent all remaining responses to ACS.
  4) ACS has received all remaining requests from CPE.

  If the above conditions have not been met and ACS has not received any response from CPE in the past 30 seconds, ACS may consider to terminate the session.

III. Our Implemented Solution

To increase the impact of our recent research [8], [15] and as well as to extend it, we have developed the *TR-069 bundle* as an universal software package for any OSGi framework [15]. In case of this particular work, we have tested this bundle together with the OSGi Knopflerfish framework [16]. The motivation to focus on the OSGi platforms follows from the fact that today's MTCGs are mostly built with pre-configured operating systems, wherever OSGi framework is used [8]. Further in this section, the key parts of the created TR-069 bundle are described.

A. Application Logic

Remote configuration of the network node consists of two building blocks: (i) ACS server and (ii) TR-069 client; the application logic is depicted in Fig. 5. Our solution is based on an open source implementation of ACS called GenieACS [17], which combines modern technologies including Mongo DB, Node.js, and Redis server. Proposed TR-069 client follows the communication logic from modus TR-069 project – originally developed in Orange Labs [21]. Modus TR-069 implements the OSGi standards [19], [20] and uses the Knopflerfish framework as a runtime environment. Comparison of available TR-069 clients is shown in Table IV. In modus TR-069 architecture, each TR-069 RPC method is implemented as a separated bundle, see Fig. 4. This approach brings modularity into implementation of the TR-069 RPC methods –

new method can be easily added as new bundle. Further, bundle must import *RPC Method Mng Service* which serves as interface between bundle and *TR69 Client API*. Mentioned TR69 Client API bundle is used as a core part of the system and enables communication between bundles and external applications. Except *RPC Method Mng Service*, modus TR-069 contains following key elements:

- **OSGi Bundle** – this service allows modus TR-069 to control (install/uninstall or start/stop) other bundles launched within OSGi framework.
- **Data Model Bundle** – defines data model used in communication between ACS and CPE. In this work, TR-106 model (Internet Gateway Device) takes place.
- **File Persist Bundle** – stores obtained configuration data into system memory for subsequent use.
- **Server Com Bundle** – this service acts as a server which controls all communication (initialization, maintaining, etc.) between ACS and CPE. Also, bundle provides FTP, TFTP and HTTP service for file downloading over the TR-069 protocol.

TABLE IV: Comparison of TR-069 CWMP client implementations [18]

RPC CPE metody		EasyCwmp	netcwmp	freecwmp	cwmplclient	open-tr069	modus-tr069
Mandatory	GetRPCMethods	🟩	🟩	🟩	🟩	🟨	🟩
	SetParameterValues	🟩	🟨	🟨	🟩	🟨	🟩
	GetParameterValues	🟩	🟨	🟨	🟩	🟨	🟩
	SetParameterAttributes	🟩	🟥	🟨	🟥	🟥	🟩
	GetParameterAttributes	🟩	🟥	🟥	🟥	🟥	🟩
	GetParameterNames	🟩	🟨	🟥	🟩	🟨	🟩
	AddObject	🟩	🟨	🟨	🟥	🟨	🟩
	DeleteObject	🟩	🟨	🟥	🟥	🟥	🟩
	Reboot	🟩	🟩	🟩	🟩	🟥	🟩
	Download	🟩	🟨	🟨	🟩	🟥	🟩
Optional	FactoryReset	🟩	🟩	🟩	🟥	🟥	🟥
	ScheduleInform	🟩	🟥	🟥	🟥	🟥	🟥
	Upload	🟥	🟩	🟥	🟩	🟥	🟩
	GetQueuedTransfers	🟥	🟥	🟥	🟥	🟥	🟥
	SetVouchers	🟥	🟥	🟥	🟥	🟥	🟥
	GetOptions	🟥	🟥	🟥	🟥	🟥	🟥

Further, obtained data is processed and visualized by the following packages: (i) Item, (ii) Core, (iii) TR069 Parser, and (iv) WebConsole.

B. Communication Logic

The application data structure is defined in *Item bundle*, see Fig. 5. This package is utilized only as a library without its own activator defining standard format of messages exchanged between the bundles. For this reason, it is necessary to import this package in each bundle communicating with Core one. As a provider of Items service (register all available items, e.g., smart meters), *Core bundle* is used. Each item is addressed by the serial number as the unique device identifier. The selected data structure, the ConcurrentHashMap, guarantees thread-save access. On the top of it, Core bundle must be started as first since it acts as an activator and control process for all others.

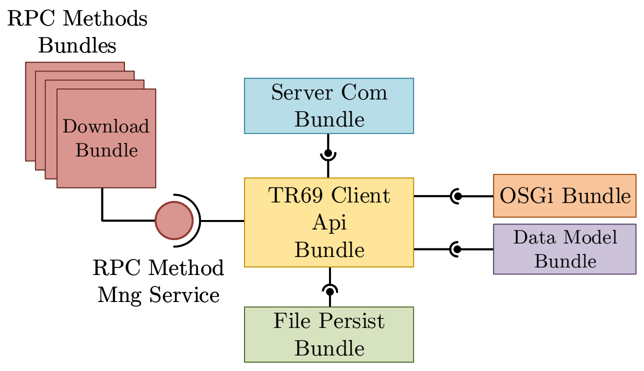


Fig. 4: Modus TR-069 Architecture

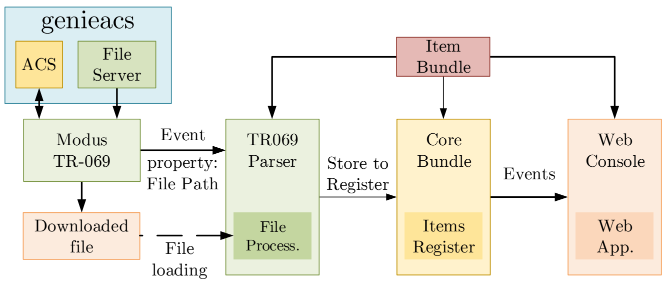


Fig. 5: Location of entities in case of using TR-069 protocol.

The main advantage of using the described model is the possibility to add new bundles (packages) to OSGi framework without the need to modify the source code in Core bundle. The only condition to be fulfilled for a new bundle is an import of Items service. This logic provides a possibility for the one way communication between all bundles and Core bundle. To resolve this issue, we have used OSGi Event Admin service allowing the backward communication between Core bundle and other packages. In this case, Core bundle is used as a source of the OSGi events that other packages are listening to, see Fig. 6. Individual events are distinguished with a dedicated array called *event topic*. Payload of each event starts by word *property* which contains one or more Item objects.

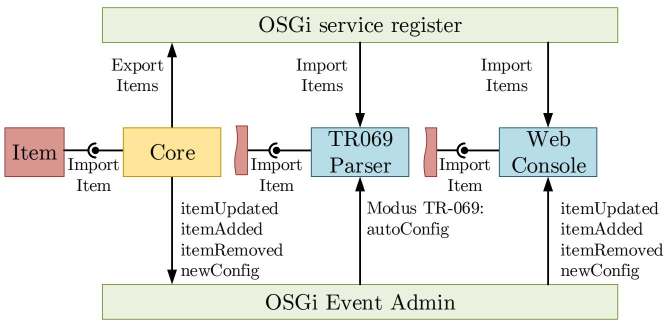


Fig. 6: Communication between Core bundle and other bundles within the OSGi Knopflerfish framework.

C. TR-069 Parser

Device configuration is carried out by the received configuration file processing – file structures may differ based on the agreed terms between ACS and CPE(s). Therefore, it is not necessary to know the file structure during its download phase, but on the other hand, it is crucial to be aware of such structure when processing on MTCG. This method is a default option for remote configuration of the network devices for telecommunication operators – we have performed the test of our solution in cooperation with Telekom Austria Group (TAG) company.

Since client of Modus TR-069 was not developed for direct cooperation with other systems, it was necessary to modify it to make the cooperation possible. Once the new configuration file has been downloaded, the TR-069 creates an event, which contains the path to this new configuration. The bundle TR069 Parser listens for this event and processes the downloaded configuration file and loads the configuration into the system.

The implemented TR-069 communication procedure is shown in Fig. 7. TR-069 protocol is used for the new configuration file notification – represented in TR-069 terminology by TR-069 configuration files and defined by number '3' as FileType array. Developed TR-069 client allows to use HTTP or FTP as transport protocol. Further, downloaded file is processed by *TR-069 Parser bundle*. Note that in this phase of the development, it supports neither secure connection nor authentication to the ACS required by some telecom providers.

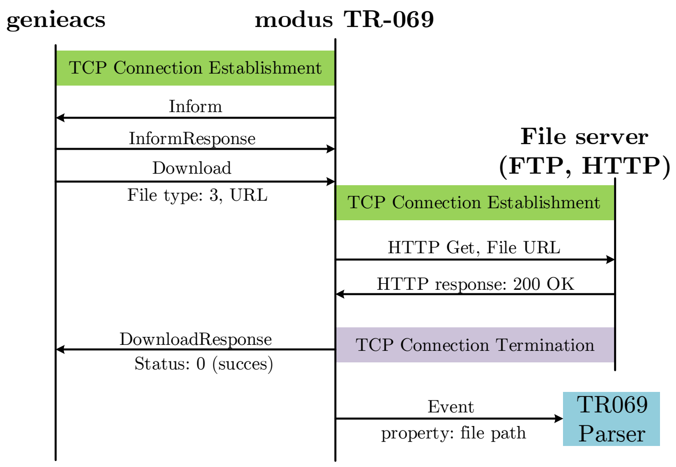


Fig. 7: Obtaining configuration file using implemented TR-069 protocol.

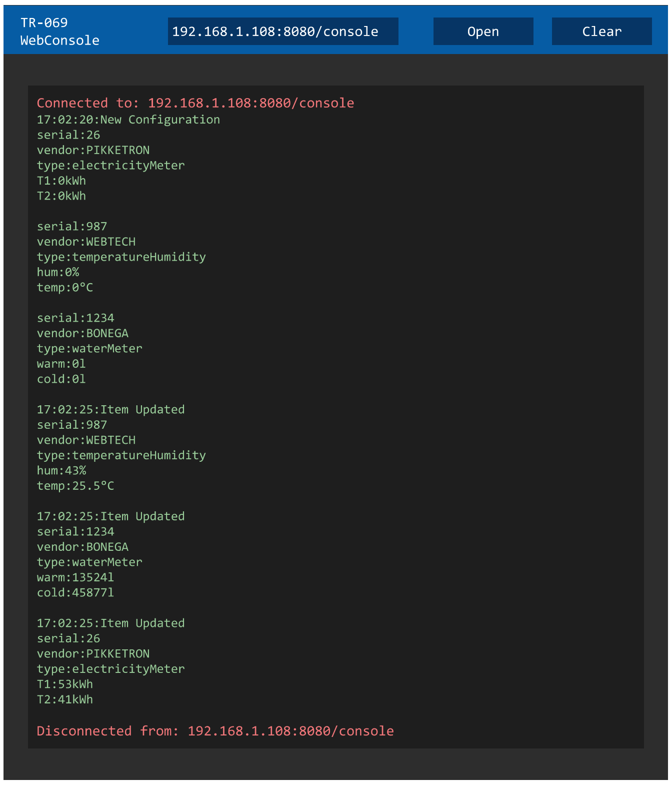
D. Tested scenario

Proposed scenarios were tested in the first phase within created local communication network. Further, the real communication infrastructure provided by Telekom Austria Group was used. Client side was realized by IP router NEC RGG200LV (based on the ARM (Acorn RISC Machine) architecture) with implemented JAVA OSGi framework – equipped with modus TR-069 client and our created application. CPE logic implementation was devoted from root node called InternetGatewayDevice, defined in TR-098 data model. Rest of the parameters used at the side of CPE were defined in DeviceInfo node, namely: (i) Manufacturer, (ii) SerialNumber, (iii) SoftwareVersion, and (iv) HardwareVersion.

First part of *configuration update* consists from file upload to ACS server. In this step, the CPE which will receive configuration update could be defined together by the parameters mentioned in previous paragraph (as an unique identifier for target (specific) device or for whole group of devices with same parameters). After successful file upload, ACS server sends notification update (Download procedure) for selected devices. These devices parse incoming message and create second communication channel to file server. In our particular scenario, unencrypted HTTP application protocol was used for data transfers. Content of transferred file is independent on TR-069 protocol. In this work, JSON (JavaScript Object Notation) file is used. File structure is considered in (i) core section of created application, and (ii) in TR069 Parser bundle.

E. Console Output

In some cases, it is not possible to display the list of running events in the system console (e.g., when OSGi framework runs as a daemon in the background). Therefore, we have created a specialized *WebConsole bundle* working as a web service and displaying system events in a web console. Communication between the bundle and the web service is realized by WebSocket protocol which is an elementary part of HTML 5. WebConsole bundle operates as OSGi EventHandler listening to all OSGi events utilized within the SyMPHOnY project, see Fig. 5. Each event is processed and the payload part is sent to the web services, and, finally displayed, see Fig. 8 where communication between ACS and CPE is shown.



```
Connected to: 192.168.1.108:8080/console
17:02:20:New Configuration
serial:26
vendor:PIKKETRON
type:electricityMeter
T1:0kWh
T2:0kWh

serial:987
vendor:WEBTECH
type:temperatureHumidity
hum:0%
temp:0°C

serial:1234
vendor:BONEGA
type:waterMeter
warm:0l
cold:0l

17:02:25:Item Updated
serial:987
vendor:WEBTECH
type:temperatureHumidity
hum:43%
temp:25.5°C

17:02:25:Item Updated
serial:1234
vendor:BONEGA
type:waterMeter
warm:13524l
cold:45877l

17:02:25:Item Updated
serial:26
vendor:PIKKETRON
type:electricityMeter
T1:53kWh
T2:41kWh

Disconnected from: 192.168.1.108:8080/console
```

Fig. 8: Console output of captured communication between ACS and CPE.

IV. Concluding Thoughts

Within the proposed logic for the remote configuration of IoT devices acting as MTCG (and MTCD devices connected to MTCG), our implementation of TR-069 protocol has demonstrated the functionality of communication between the ACS server and the end-device (CPE) in a real network.

We have successfully tested the developed solution in cooperation with Telekom Austria Group. As we aimed our solution to be universal for various types of MTCG devices, we have constructed TR-069 bundle to be compliant with the well-known OSGi frameworks. As mentioned in Section III-C, the developed TR-069 implementation in its current version does not support secure connection and authentication to the ACS. Therefore, as a next step, we are planning to implement this functionality in our TR-069 bundle.

Our main and most essential learning while working with the TR-069 protocol is such that the structure of the configuration file is not static. Following the specifics of the concrete mobile network (ACS server configuration), the configuration files may differ. In our trial, we have utilized the JSON structure [22] implemented in live A1 cellular network.

At the end of this paper, we recall that this manuscript represents extended version of our previous work – originally presented at TSP 2016 conference [9].

References

[1] M. S. Whitcup and K. LaMattina, *Juniper What is Inhibiting Growth in the Medical Device Wearable Market?* Available from: http://bit.ly/1Dfffbf, September 2014.

[2] Cisco, *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2014-2019*, February 2015.

[3] P. Masek, J. Hosek, D. Kovac, F. Kropfl, *M2M Gateway: The Centrepiece of Future Home.* in Proc. of 6th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT). St. Petersburg, Russia. pp. 286–293. 2014.

[4] M. Gerasimenko, V. Petrov, O. Galinina, S. Andreev, Y. Koucheryavy, *Energy and delay analysis of LTE-advanced RACH performance under MTC overload.* Globecom Workshops (GC Wkshps), IEEE. pp. 1632–1637. 2012.

[5] P. Masek, K. Zeman, Z. Kuder, J. Hosek, S. Andreev, R. Fujdiak, F. Kropfl, *Wireless M-BUS: An Attractive M2M Technology for 5G-Grade Home Automation.* in Proc. of EAI International Conference on CYber physiCaL systems, iOt and sensors Networks (CYCLONE). pp. 1–12. ISBN: 978-1-4673-9282-2. 2015.

[6] N. Himayat, S.-P. Yeh, A. Y Panah, S. Talwar, M. Gerasimenko, S. Andreev, Y. Koucheryavy, *Multi-radio heterogeneous networks: Architectures and performance.* in Proc. of International Conference on Computing, Networking and Communications (ICNC). pp. 252–258. 2014.

[7] O. Galinina, S. Andreev, M. Gerasimenko, Y. Koucheryavy, N. Himayat, S.-P. Yeh, S. Talwar, *Capturing spatial randomness of heterogeneous cellular/WLAN deployments with dynamic traffic.* Journal on Selected Areas in Communications. vol. 32. issue 6. pp. 1083–1099. 2014.

[8] GitHub: SyMPHOnY (Smart Multi-Purpose Home Gateway). Available from: https://github.com/SyMPHOnY-/Smart-Home-Gateway/wiki/

[9] M. Stusek, P. Masek, D. Kovac, A. Ometov, J. Hosek, F. Kropfl, S. Andreev, *Remote Management of Inteligent Devices: Using TR-069 Protocol in IoT.* in Proc. of the 39th International Conference on Telecommunication and Signal Processing, TSP 2016. Vienna, Austria. pp. 1–5. ISBN: 978-1-5090-1287-9. 2016.

[10] A. Ometov, S. Andreev, A. Turlikov, Y. Koucheryavy, *Characterizing the effect of packet losses in current WLAN Deployments.* in Proc. of 13th International Conference on ITS Telecommunications (ITST). pp. 331–336. IEEE. 2013.

[11] S. Andreev, P. Gonchukov, N. Himayat, Y. Koucheryavy, A. Turlikov, *Energy efficient communications for future broadband cellular networks.* Computer Communications, Volume 35, Issue 14, pp. 1662–1671. 2012.

[12] J. Bernstein, T. Spets, *CPE WAN Management Protocol.* DSL Forum, Tech. Rep. TR-069. 2004.

[13] D. Dasgupta, S. Saha, A. Negatu, *Techniques for validation and controlled execution of processes, codes and data: A survey,* in Proc. of International Conference on Security and Cryptography (SECRYPT). pp. 1–9. 2010.

[14] Incognito, *Broadband Forum TR-069 standards support.* [online]. Available from: http://bit.ly/1OdCsjM/

[15] M. Stusek, J. Hosek, D. Kovac, P. Masek, P. Cika, J. Masek, F. Kropfl, *Performance Analysis of the OSGi-based IoT Frameworks on Restricted Devices as Enablers for Connected-Home.* in Proc. of 7th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT). Brno, Czech Republic. pp. 211–216. ISBN: 978-1-4673-9282-2. 2015.

[16] Knopflerfish, *Open Source OSGi SDK.* [Online]. Available from: http://www.knopflerfish.org/

[17] GenieACS, *Smart, fast TR-069 ACS.* [online]. Available from: https://genieacs.com/

[18] A. Stack. TR-069 cwmp client implementation: open sources comparison. In: *Stackoverflow* [online]. 2014. Available from: http://bit.ly/1PTzzWS

[19] OSGi, *OSGi Alliance.* [Online]. Available from: http://www.osgi.org/

[20] R. Hall, K. Pauls, S. McCulloch, D. Savage, *OSGi in action: creating modular applications in Java.* Greenwich [Conn.]: Manning. 548 p. ISBN: 19-339-8891-6. 2012.

[21] France Telecom, *Modus TR069.* [online]. Available from: http://modus-tr-069.sourceforge.net/

[22] A. Greenspan, L. Cameron, *Monetize the Internet of Things: JSON turns a flood of data into business actions and results.* 2015 [online]. Available from: http://bit.ly/1He09Vu/